Nonlinear Dimensionality Reduction:

Kernel PCA and Manifold Learning

Assoc. Prof. Karl Ezra Pilario, Ph.D.

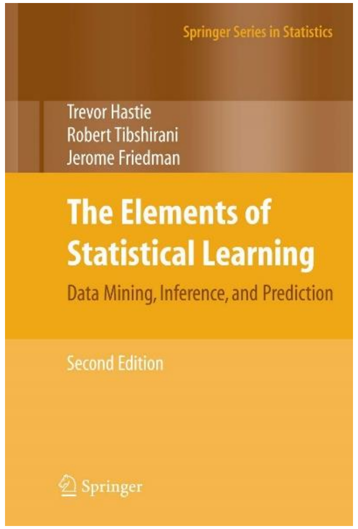
Process Systems Engineering Laboratory
Department of Chemical Engineering
University of the Philippines Diliman

Outline

- Kernel Methods
  - Kernel PCA

- Manifold Learning
  - Multidimensional Scaling
  - Isomap Embedding
  - Locally Linear Embedding (LLE)
  - Laplacian Eigenmaps
  - t-Distributed Stochastic Neighborhood Embedding (t-SNE)
  - Uniform Manifold Approximation and Projection (UMAP)

Hastie *et al.* (2008)
The Elements of Statistical Learning.
2nd Ed. Springer.



Springer Series in Statistics

Trevor Hastie
Robert Tibshirani
Jerome Friedman

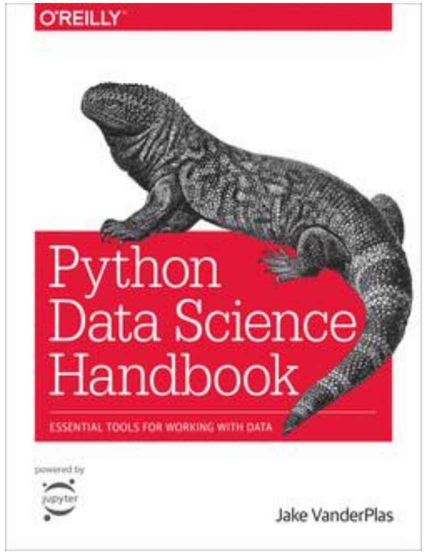
The Elements of Statistical Learning

Data Mining, Inference, and Prediction

Second Edition

Springer

Jake VanderPlas (2016)
Python Data Science Handbook.
O’Reilly Media, Inc.



O'REILLY

Python Data Science Handbook

ESSENTIAL TOOLS FOR WORKING WITH DATA

powered by jupyter

Jake VanderPlas

Recall: PCA

- Previously, we discussed PCA as a popular dimensionality reduction method.
- PCA gives a new set of uncorrelated features with maximum variance.
- We learned that the PCA algorithm simply amounts to the eigenvalue decomposition of the sample covariance matrix.

PCA Algorithm

1. Standardize the Data (zero-mean, unit-variance)
2. Compute the covariance of $\boldsymbol{X}$: $$\boldsymbol{\Sigma} = \frac{1}{N-1}\boldsymbol{X}^T\boldsymbol{X}$$
3. Compute the eigenvalue decomposition of $\boldsymbol{\Sigma}$: $$\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$
4. Choose only n principal components, then get $\boldsymbol{Y}$: $$\boldsymbol{P} = \boldsymbol{V}_n$$ $$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{P}$$

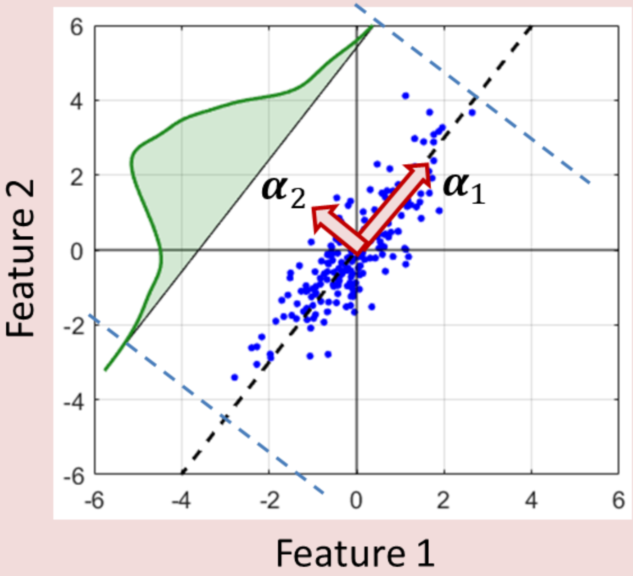
The goal of PCA is to find a projection matrix, $\boldsymbol{P} \in \Re^{m\times m}$, such that $\boldsymbol{P}$ is orthonormal and the *variance* of the projected data, $\boldsymbol{Y} \in \Re^{N\times m}$, is maximized:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{P}$$

where: $\boldsymbol{Y} = [\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad \dots \quad \boldsymbol{y}_m]$ $\boldsymbol{y}$'s are called scores.

$\boldsymbol{X} = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \dots \quad \boldsymbol{x}_m]$

$\boldsymbol{P} = [\boldsymbol{\alpha}_1 \quad \boldsymbol{\alpha}_2 \quad \dots \quad \boldsymbol{\alpha}_m]$ $\boldsymbol{\alpha}$'s are called loadings / coefficients.



$(\boldsymbol{y}_1, \boldsymbol{\alpha}_1)$ is the 1st principal component.
$(\boldsymbol{y}_2, \boldsymbol{\alpha}_2)$ is the 2nd principal component.
...and so on...

Why does PCA accomplish dimensionality reduction?

After PCA, we can take only the *first few* scores as the new extracted features, then discard the rest.

$$\begin{matrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\\ \vdots\\ \boldsymbol{x}_m\end{matrix} \Longrightarrow \begin{matrix}\boldsymbol{y}_1 & \text{(PC1)}\\ \boldsymbol{y}_2 & \text{(PC2)}\end{matrix}$$

© 2023 Karl Ezra S. Pilario, PhD.

Kernel PCA

In Kernel PCA (KPCA), the goal is the same: maximize variance using orthogonal basis projections. But KPCA can employ nonlinear projections due to the kernel trick!

PCA Algorithm

1. Standardize the Data (zero-mean, unit-variance)
2. Compute the covariance of $\boldsymbol{X}$: $$\boldsymbol{\Sigma} = \frac{1}{N-1}\boldsymbol{X}^T\boldsymbol{X}$$
3. Compute the eigenvalue decomposition of $\boldsymbol{\Sigma}$: $$\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$
4. Choose only n principal components, then get $\boldsymbol{Y}$: $$\boldsymbol{P} = \boldsymbol{V}_n$$ $$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{P}$$

Kernel PCA Algorithm

1. Standardize the Data (zero-mean, unit-variance)
2. Choose a kernel function (e.g. RBF): $$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{\sigma}\right)$$
3. Compute the covariance as a kernel matrix, $\boldsymbol{K}$, using the kernel k: $$\mathbf{K} = \left[k(\boldsymbol{x}_i, \boldsymbol{x}_j)\right]$$
4. Center the kernel matrix: $$\mathbf{K}_c = \boldsymbol{K} - \mathbf{1}_N\boldsymbol{K} - \boldsymbol{K}\mathbf{1}_N + \mathbf{1}_N\boldsymbol{K}\mathbf{1}_N$$
5. Compute the eigenvalue decomposition of $\mathbf{K_c}$: $$\mathbf{K}_c = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$
6. Choose only n principal components: $$\boldsymbol{P} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{V}_n$$ $$\boldsymbol{Y} = \boldsymbol{P}^T\boldsymbol{K}_c$$

   To project any new test sample, $\boldsymbol{x}'$: $$\boldsymbol{y} = \boldsymbol{P}^T\boldsymbol{k}(\boldsymbol{x}', \boldsymbol{X})$$

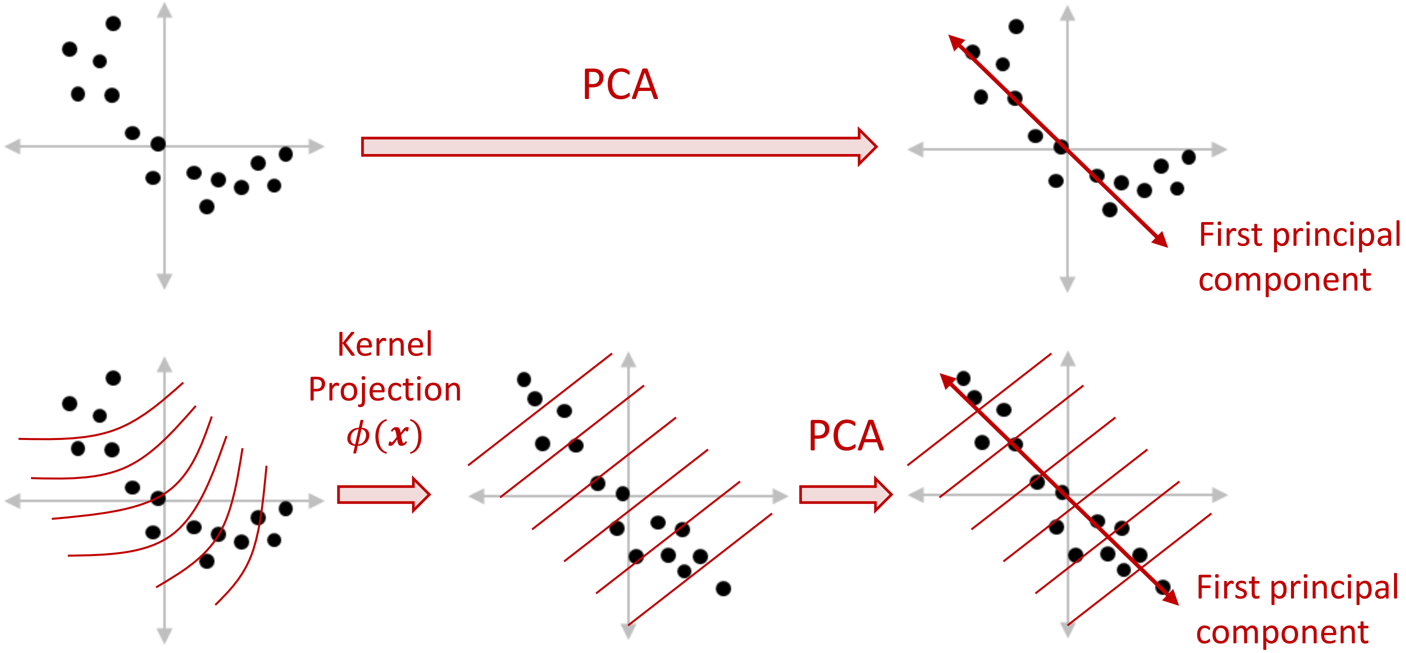
Note: $\mathbf{1}_N$ is an $N \times N$ matrix, with elements $(\mathbf{1}_N)_{ij} = 1/N$.

Kernel PCA

In Kernel PCA (KPCA), the goal is the same: maximize variance using orthogonal basis projections. But KPCA can employ nonlinear projections due to the kernel trick!

Principal Components Analysis

Kernel Principal Components Analysis



PCA

First principal component

Kernel Projection $\phi(\boldsymbol{x})$

PCA

First principal component

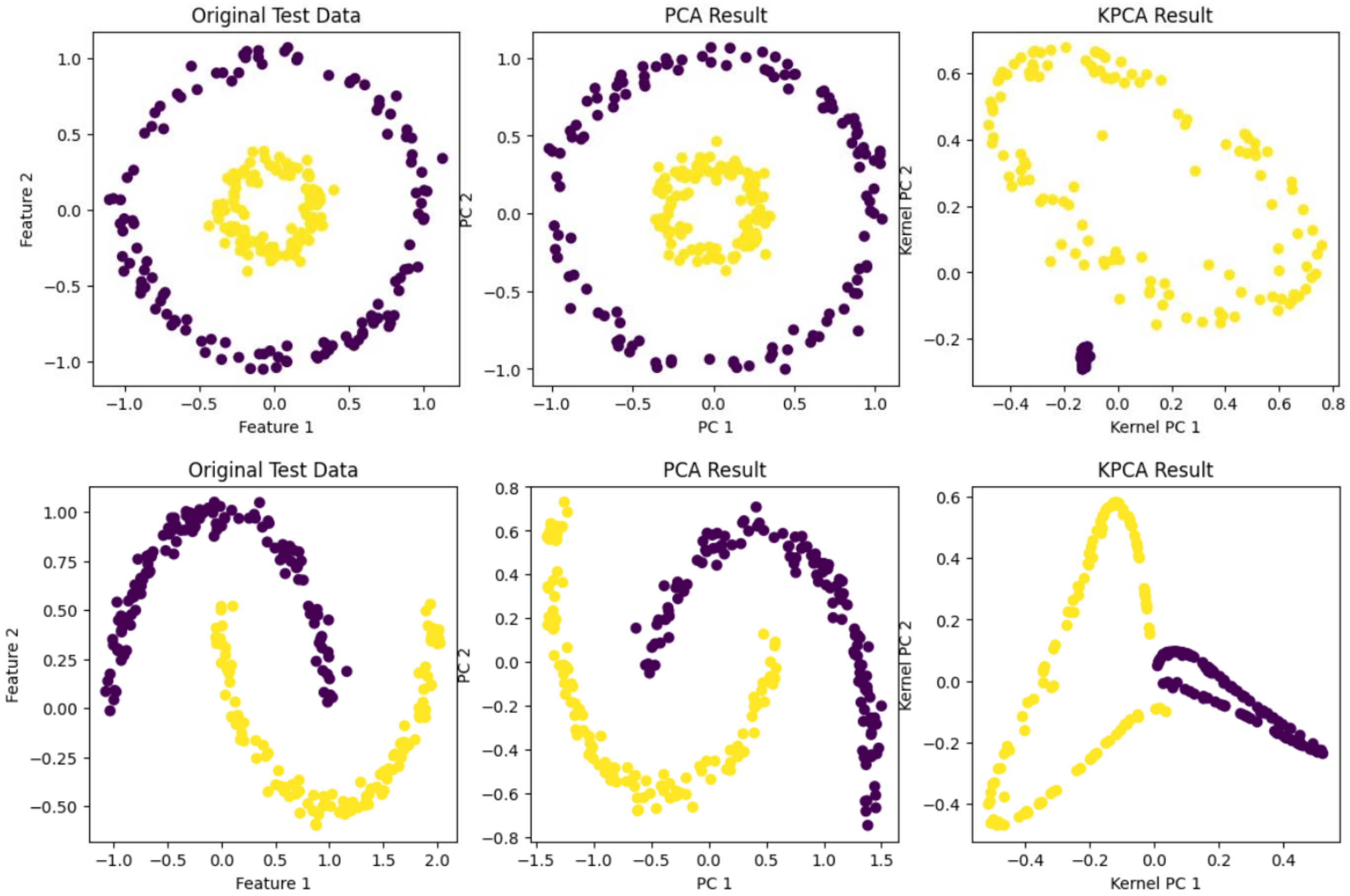
- The PCA covariance is computed as: $$\boldsymbol{\Sigma} = \frac{1}{N-1}\boldsymbol{X}\boldsymbol{X}^T$$
- The KPCA covariance is approximated by the kernel matrix: $$\boldsymbol{\Sigma} = \frac{1}{N-1}\boldsymbol{\phi}(\boldsymbol{X})\boldsymbol{\phi}(\boldsymbol{X})^T = \mathbf{K} = \left[k(\boldsymbol{x}_i, \boldsymbol{x}_j)\right]$$

Kernel PCA

In Kernel PCA (KPCA), the goal is the same: maximize variance using orthogonal basis projections. But KPCA can employ nonlinear projections due to the kernel trick!

Example 1: Concentric Circles Data and Two Moons Data

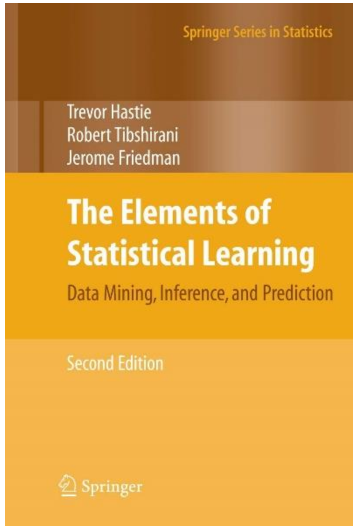
Generate 1000 samples of the concentric circles data (make_circles) and two moons data (make_moons). Split them into 75% training and 25% testing. Apply PCA and KPCA (RBF, gamma=10) on the training data, then transform the test data. Compare the results.



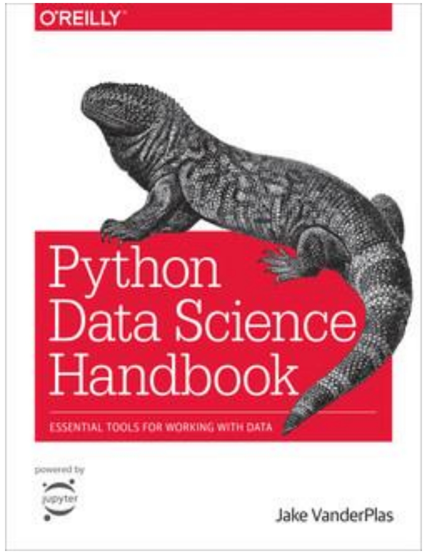
Outline

- Kernel Methods
  - Kernel PCA
- **Manifold Learning**
  - Multidimensional Scaling
  - Isomap Embedding
  - Locally Linear Embedding (LLE)
  - Laplacian Eigenmaps
  - t-Distributed Stochastic Neighborhood Embedding (t-SNE)
  - Uniform Manifold Approximation and Projection (UMAP)

Hastie *et al.* (2008)
The Elements of Statistical Learning.
2nd Ed. Springer.


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

Jake VanderPlas (2016)
Python Data Science Handbook.
O’Reilly Media, Inc.


O'REILLY
Python Data Science Handbook
ESSENTIAL TOOLS FOR WORKING WITH DATA
powered by jupyter
Jake VanderPlas

Manifold Learning

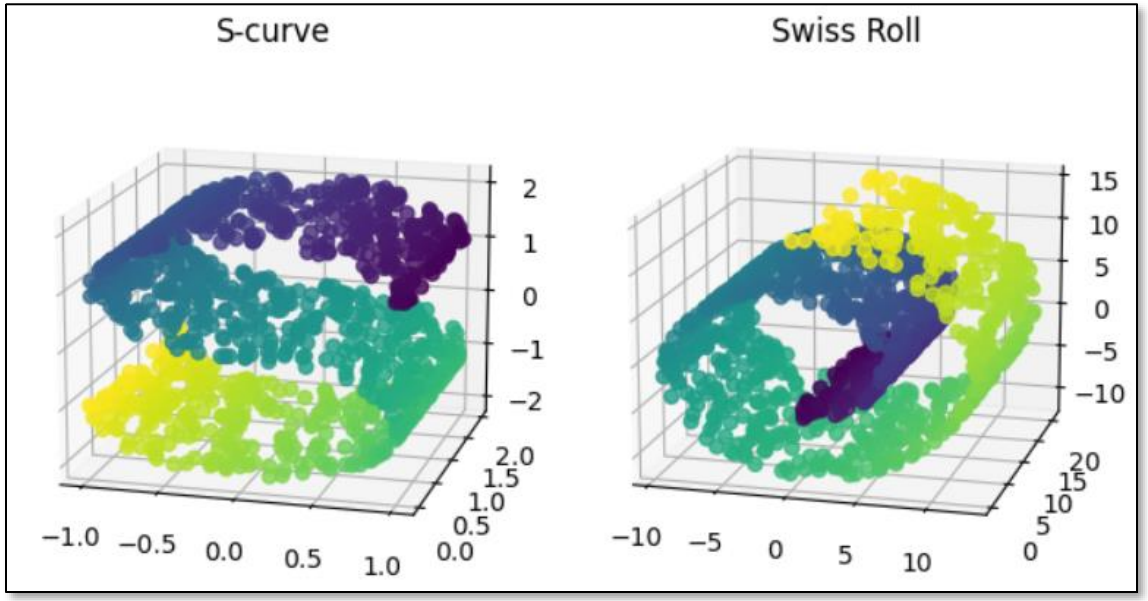
Manifold learning algorithms and KPCA are generalizations of PCA. They seek low-dimensional manifolds embedded in high-dimensional spaces.

(Python Data Science Handbook, Jake VanderPlas, 2016)

Notation:

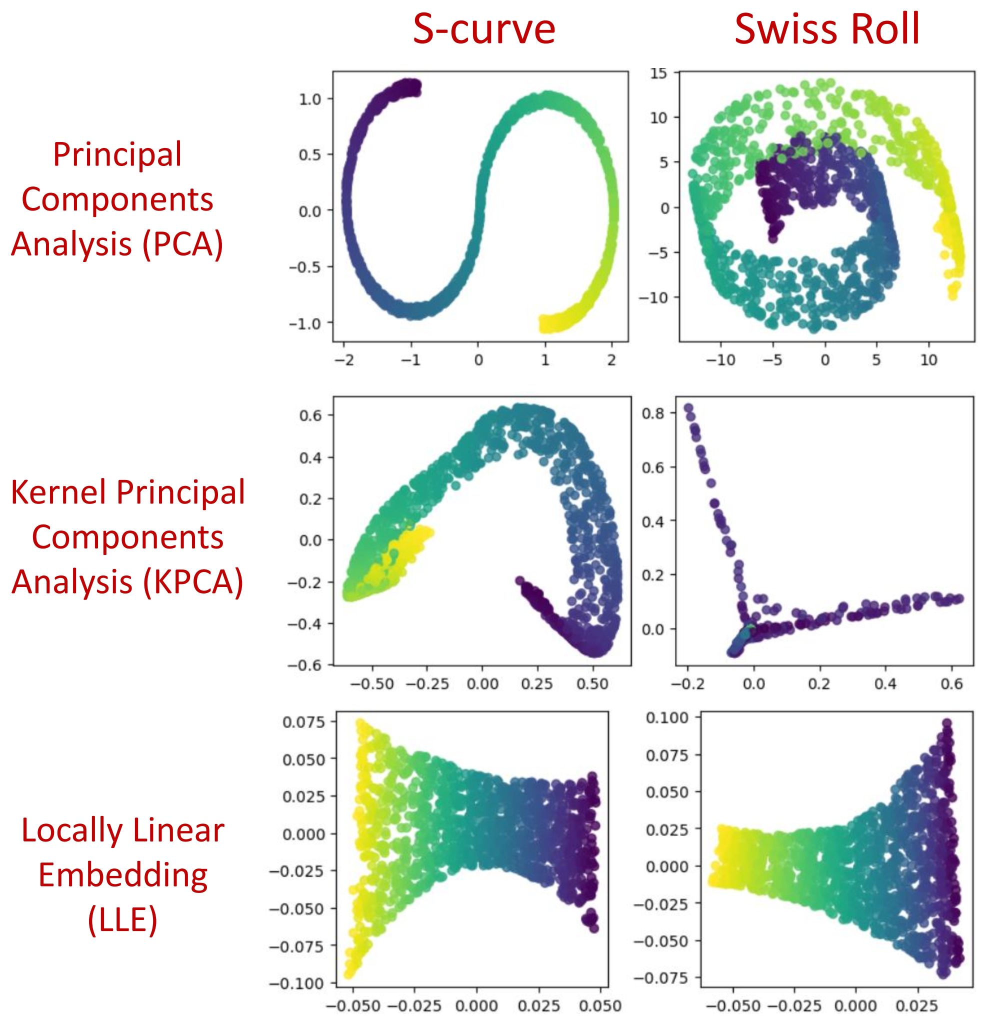
Latent Space

$$\boldsymbol{x}_i \in \Re^m \xrightarrow[\text{embedding}]{} \boldsymbol{y}_i \in \Re^d,\; d < m$$



The **S-curve** and **Swiss Roll** data sets are two popular examples that illustrate the weakness of PCA and KPCA, and the strength of other manifold learning algorithms for nonlinear dimensionality reduction.

2-D Embedding of the Data Sets



Principal Components Analysis (PCA)

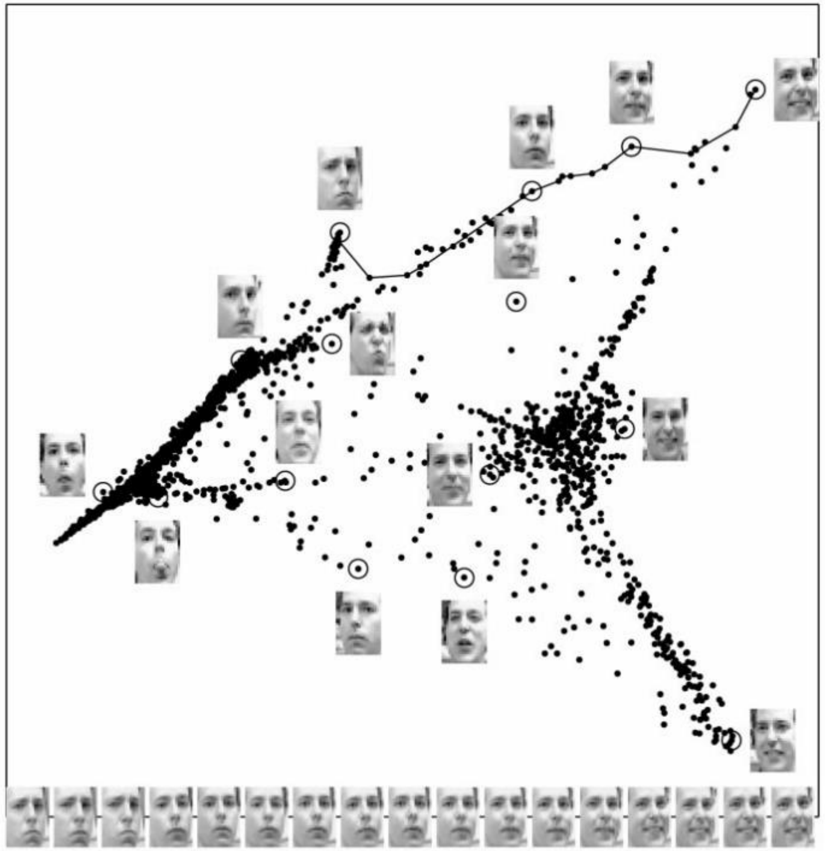
Kernel Principal Components Analysis (KPCA)

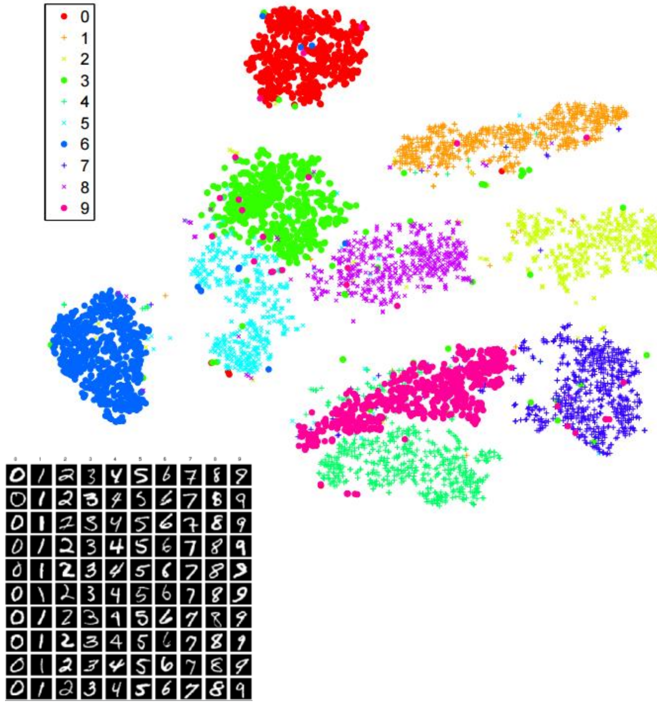
Locally Linear Embedding (LLE)

Manifold Learning

Manifold learning algorithms have many applications.

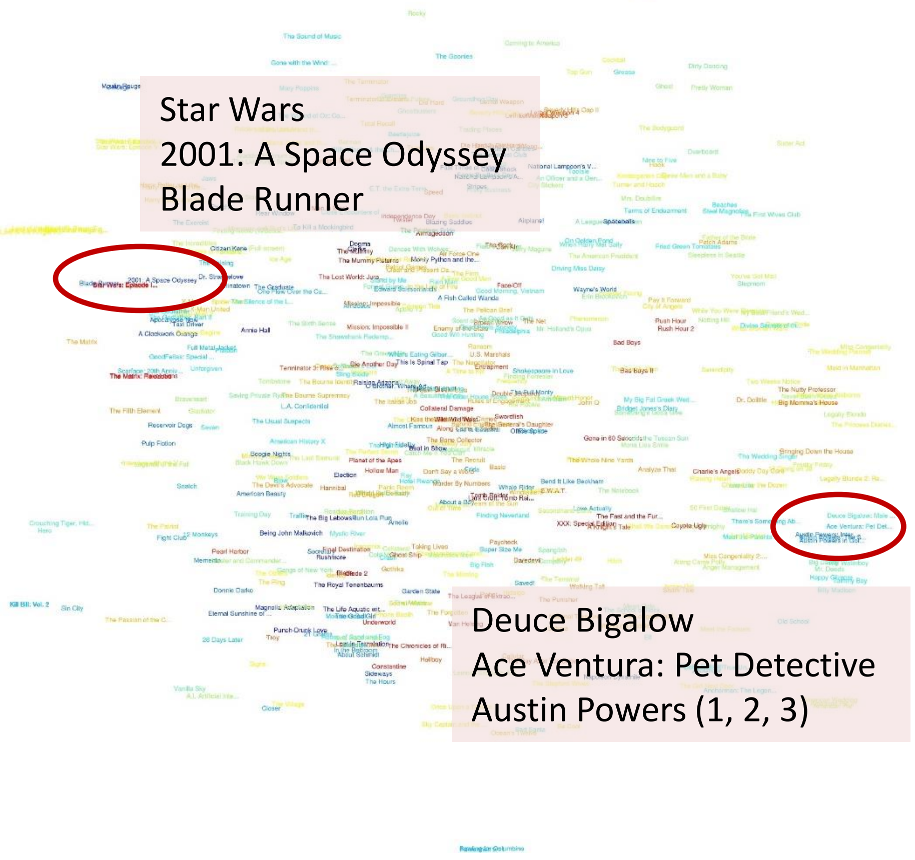
LLE applied to face images can reveal paths of varying pose and expression. (Roweis, 2000)





MNIST Handwritten Digits meaningfully projected in 2-D using t-SNE. (van der Maaten, 2008)

t-SNE visualization of 17,000 Netflix movie titles based on 10^8 user ratings.
https://lvdmaaten.github.io/tsne/examples/netflix_tsne.jpg



Star Wars
2001: A Space Odyssey
Blade Runner

Deuce Bigalow
Ace Ventura: Pet Detective
Austin Powers (1, 2, 3)

MDS and Isomap

Kruskal (1964) and Tenenbaum et al. (2000)

Multidimensional Scaling (MDS)

Related Methods: Non-metric MDS, Sammon Mapping

Step 1. Calculate pairwise distances between all samples.

$$d_{i,j} = \text{distance between samples } \boldsymbol{x}_i \text{ and } \boldsymbol{x}_j$$

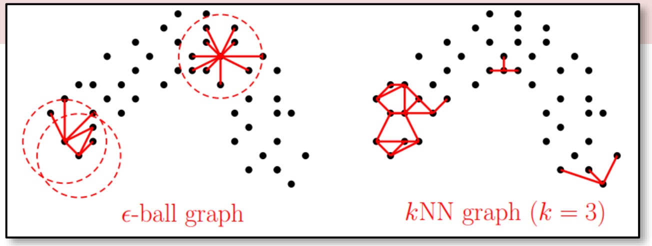
$$\boldsymbol{D} = \begin{bmatrix} d_{1,1} & d_{1,2} & d_{1,3} & \cdots & d_{1,N} \\ d_{2,1} & d_{2,2} & d_{2,3} & \cdots & d_{2,N} \\ d_{3,1} & d_{3,2} & d_{3,3} & \cdots & d_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{N,1} & d_{N,2} & d_{N,3} & \cdots & d_{N,N} \end{bmatrix}$$

(Dissimilarity matrix)

Step 2. Find low-dimensional vectors $\boldsymbol{y}_i$ such that the pairwise distances are *preserved as much as possible*. Use gradient descent.

$$\boxed{\min_{\boldsymbol{y}} \sum_{i<j} \left(\lVert \boldsymbol{y}_i - \boldsymbol{y}_j \rVert - d_{i,j} \right)^2}$$

Isomap



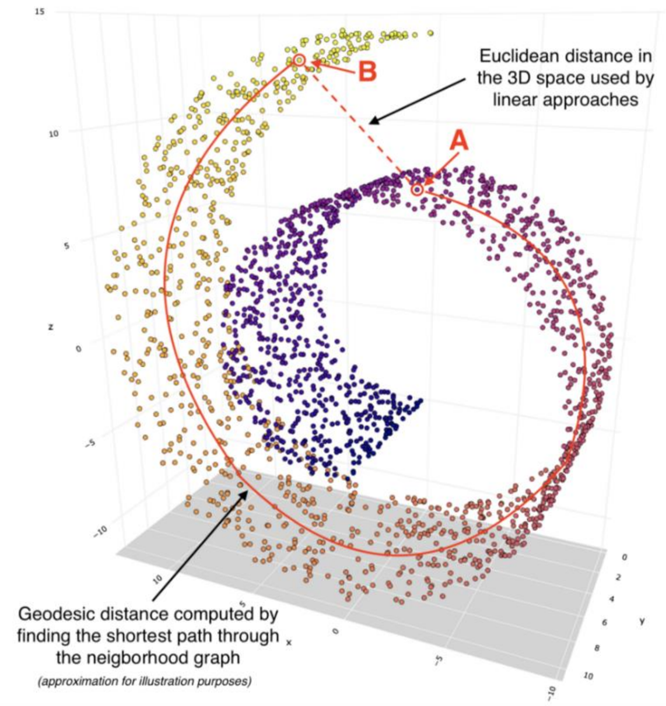
Step 1. Find all k-nearest neighbors of all samples.

Step 2. Construct a neighborhood graph: $G(V, E)$. Two samples are connected by an edge if and only if they are *neighbors*.

Step 3. Compute the all-pairs shortest path on graph G, then create a dissimilarity matrix. Use Floyd-Warshall's or Dijkstra's algorithm.

Step 4. Find low-dimensional vectors $\boldsymbol{y}_i$ such that the pairwise distances are *preserved as much as possible*. Use gradient descent.

$$\boxed{\min_{\boldsymbol{y}} \sum_{i<j} \left(\lVert \boldsymbol{y}_i - \boldsymbol{y}_j \rVert - d_{i,j} \right)^2}$$



Euclidean distance in the 3D space used by linear approaches

Geodesic distance computed by finding the shortest path through the neigborhood graph

(approximation for illustration purposes)

MDS → Euclidean distance
Isomap → Geodesic distance

Source: https://towardsdatascience.com/isomap-embedding-an-awesome-approach-to-non-linear-dimensionality-reduction-fc7efbca47a0
https://www.sjsu.edu/faculty/guangliang.chen/Math253S20/lec12laplacian.pdf

Side Note: Similarity Measures, Distances, and Metrics

The notion of *distance* between samples (objects) is important in many machine learning models.

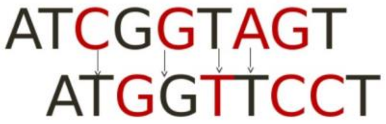
A distance d is properly called a **metric** if the following holds:

- $d(x,x) = 0$
- If $x \neq y$, then $d(x,y) > 0$ (Positivity)
- $d(x,y) = d(y,x)$ (Symmetry)
- $d(x,z) \leq d(x,y) + d(y,z)$ (Triangle inequality)

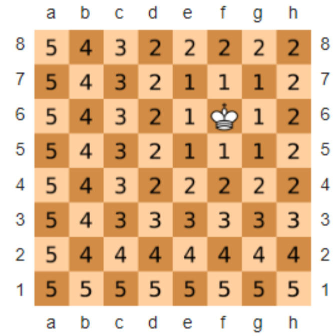
Metric Learning

Techniques that aim to automatically construct task-specific distance metrics from data in a machine learning manner.

Hamming Distance,
Edit Distance,
Levenshtein Distance

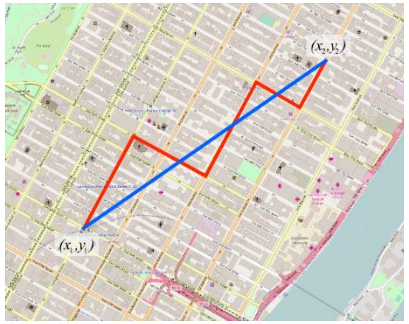


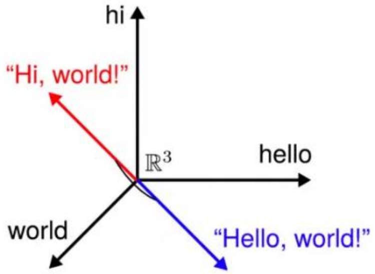
Chebyshev Distance



	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1	♔	1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

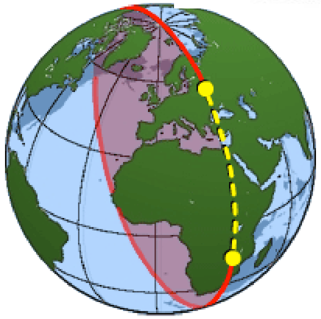
Euclidean vs. Manhattan

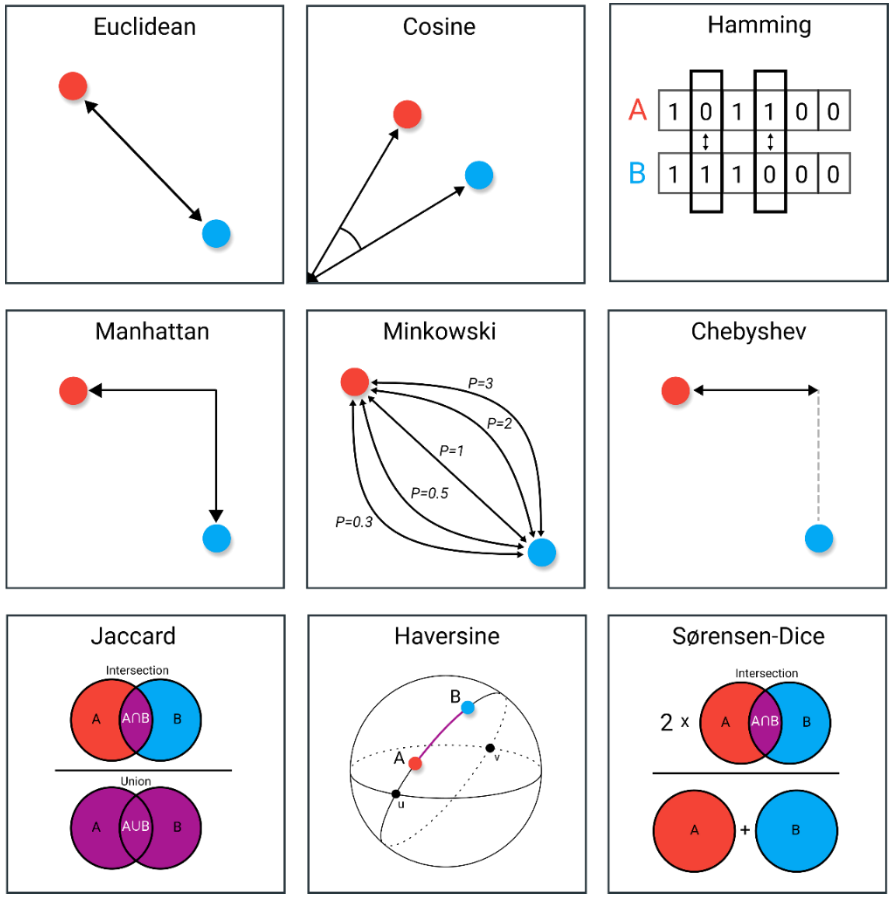




Cosine Similarity

Great Circle Distance





https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa

Locally Linear Embedding

Roweis et al. (2000)

Step 1. Find all *k*-nearest neighbors of all samples.

Step 2. Find weights w_{ij} that reconstruct each sample $\boldsymbol{x}_i$ from its neighbors.

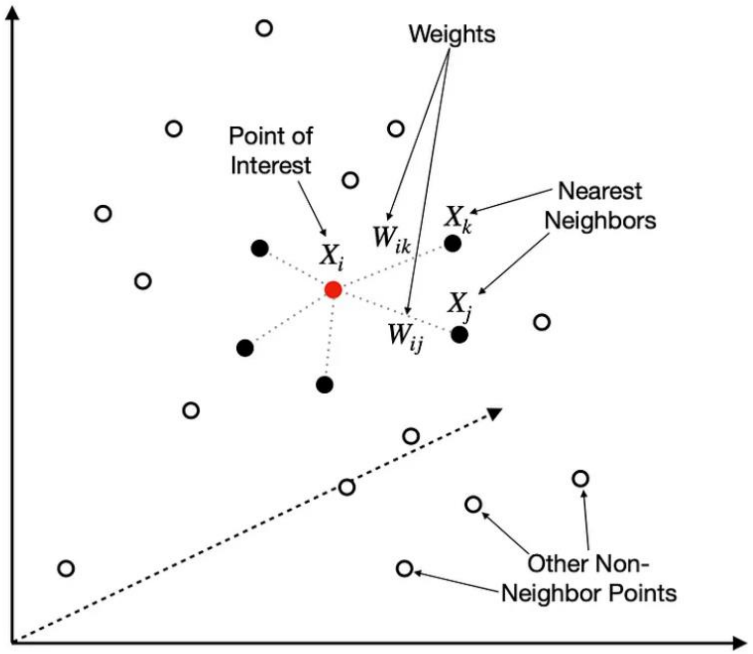
$$\boxed{\begin{aligned}&\min_{w_{ij}} \sum_i \left[\boldsymbol{x}_i - \sum_j w_{ij}\boldsymbol{x}_j\right]^2\\ \text{s.t.}\quad &\textstyle\sum_j w_{ij} = 1 \text{ for neighbors of sample } i\\ &w_{ij} = 0 \quad\text{ for non-neighbors of sample } i\end{aligned}}$$

Step 3. Find vectors $\boldsymbol{y}_i$ that retain the reconstruction weights w_{ij}, but lie in low-dimensional space.

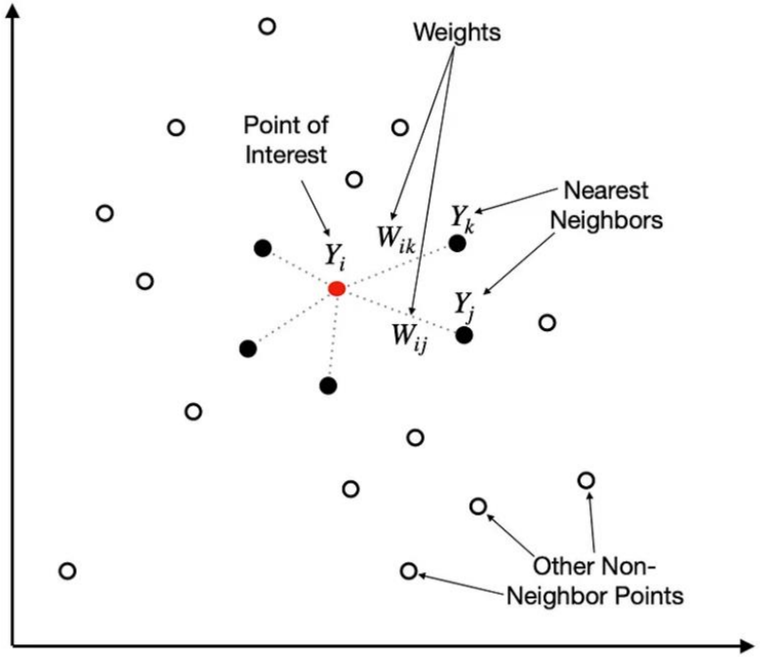
$$\boxed{\min_{y} \sum_i \left[\boldsymbol{y}_i - \sum_j w_{ij}\boldsymbol{y}_j\right]^2}$$

Note: Both optimization problems are convex:
Step 2 can be solved by a system of linear equations;
Step 3 can be solved by eigenvalue decomposition.

Original high-dimensional space



New low-dimensional space



Variants of LLE:

- **Modified LLE** – adds a regularization parameter for cases when the number of neighbors is greater than the number of components.
- **Hessian LLE (HLLE)** – adds regularization via a local Hessian estimator.
- **LTSA (Local Tangent Space Alignment)** – replaces Steps 2 and 3 with tangent space alignment.

Source: https://towardsdatascience.com/lle-locally-linear-embedding-a-nifty-way-to-reduce-dimensionality-in-python-ab5c38336107

Laplacian Eigenmaps

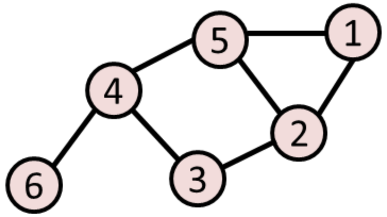
a.k.a. Spectral Embedding. *Belkin and Niyogi (2003)*

Definition: Graph Laplacian

Given a simple weighted graph, $G(V, E, \boldsymbol{W})$, having n vertices, the graph Laplacian matrix $\boldsymbol{L}$ is defined as:

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \in \Re^{n\times n}$$

where: $\boldsymbol{D}$ = degree matrix
$\boldsymbol{W}$ = *weights matrix* or *adjacency matrix* if graph is unweighted.



Example:

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Degree matrix Adjacency matrix

- Intuitively, the eigenvalues of $\boldsymbol{L}$ are measures of *how well a graph is connected*.[1] The smallest non-zero eigenvalue of $\boldsymbol{L}$ is called the *Fiedler value*.
- The smallest eigenvalue of $\boldsymbol{L}$ is always 0 and the corresponding eigenvector is $\mathbf{1} = [1, 1, 1, \dots]^T$, i.e. $\boldsymbol{L}\mathbf{1} = \mathbf{0}$.
- The number of zero eigenvalues of $\boldsymbol{L}$ is equal to the number of connected components in G.

Laplacian Eigenmaps aim to find $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ that solves:

$$\min \frac{1}{2}\sum_{ij} w_{ij}\left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^2$$

Interpretation:

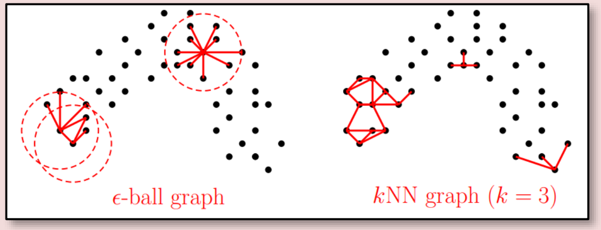
If $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close to each other in the original space, then w_{ij} (the i,j-th element of $\boldsymbol{W}$) is close to 1 and the minimizer forces $\boldsymbol{y}_i$ to be close to $\boldsymbol{y}_j$ in the reduced space. Similarly, if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart, then w_{ij} tends to 0 and the minimizer tends to *not care* where $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are mapped in the reduced space.

Where does the Graph Laplacian come in?

Well, it turns out that:

$$\boxed{\frac{1}{2}\sum_{ij} w_{ij}\left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^2 = \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}}$$

Step 1. Construct a neighborhood graph, $G(V, E)$.



Step 2. Assign weights, $\boldsymbol{W} \in \Re^{n\times n}$, to the graph edges:

Option 1: $\boldsymbol{W}$ = adjacency matrix (0/1)

Option 2: $\boldsymbol{W} = \exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$ (Heat kernel)

Step 3. Compute the graph Laplacian: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$

Step 4. Solve the eigenvalue problem: $\boldsymbol{L}\mathbf{v} = \lambda\boldsymbol{D}\mathbf{v}$

- Ignore the eigenvector $\mathbf{v}_1$ with the $\lambda = 0$ eigenvalue.
- The coordinates in reduced space are given by the eigenvectors corresponding to the smallest eigenvalues:

$$\boldsymbol{Y} = [\mathbf{v}_2, \mathbf{v}_3, \dots \mathbf{v}_\mathrm{d}] \in \Re^{n\times d}$$

[1] https://www.johndcook.com/blog/2016/01/07/connectivity-graph-laplacian

Side Note: The Rayleigh Quotient

Many embedding algorithms rely on eigenvalue decomposition: PCA, KPCA, LDA, Laplacian Eigenmaps.
These eigenvalue problems are all related to the Rayleigh Quotient.

Definition: Rayleigh Quotient

The Rayleigh Quotient for a fixed symmetric matrix $\boldsymbol{A} \in S^n\ (\Re)$ is a multivariate function $f: \Re^n - \{\mathbf{0}\} \rightarrow \Re$ defined by:

$$f(\boldsymbol{x}) = \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

Definition: Generalized Rayleigh Quotient

For a fixed symmetric matrix $\boldsymbol{A} \in S^n\ (\Re)$ and a positive definite matrix $\boldsymbol{B}$ of the same size, a generalized Rayleigh Quotient corresponding to them is given by $f: \Re^n - \{\mathbf{0}\} \rightarrow \Re$ defined by:

$$f(\boldsymbol{x}) = \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{B} \boldsymbol{x}}$$

Theorem

If a given symmetric matrix $\boldsymbol{A} \in S^n\ (\Re)$ has λ_1 and λ_n as its largest and smallest eigenvalues, with associated eigenvectors, $\mathbf{v}_1$ and $\mathbf{v}_n$, then the **global maximum and minimum of its Rayleigh Quotient** are exactly λ_1 and λ_n, respectively, and they occur at $\boldsymbol{x} = \pm\mathbf{v}_1$ and $\boldsymbol{x} = \pm\mathbf{v}_n$, respectively.

$$\max_{\boldsymbol{x}\neq\mathbf{0}} \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}} = \max_{\substack{\boldsymbol{x}\neq\mathbf{0},\\ \boldsymbol{x}^T\boldsymbol{x}=1}} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} = \lambda_1 \ @ \ \boldsymbol{x} = \pm\mathbf{v}_1$$

$$\min_{\boldsymbol{x}\neq\mathbf{0}} \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}} = \min_{\substack{\boldsymbol{x}\neq\mathbf{0},\\ \boldsymbol{x}^T\boldsymbol{x}=1}} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} = \lambda_n \ @ \ \boldsymbol{x} = \pm\mathbf{v}_n$$

Proof: The same way we derived the PCA projection matrix.

The Lagrangian can be written as:

$$\max_{\substack{\boldsymbol{x}\neq\mathbf{0}\\ \boldsymbol{x}^T\boldsymbol{x}=1}} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \quad \Longrightarrow \quad \mathcal{L}(\boldsymbol{x}, \lambda) = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} - \lambda(\boldsymbol{x}^T \boldsymbol{x} - 1)$$

Taking the partial derivative w.r.t. $\boldsymbol{x}$:

Definition of the Eigenvalue

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = 2\boldsymbol{A}\boldsymbol{x} - \lambda(2\boldsymbol{x}) = \mathbf{0} \quad \Longrightarrow \quad \boxed{\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}}$$

Source: https://www.sjsu.edu/faculty/guangliang.chen/Math250/lec4RQ.pdf

Side Note: The Rayleigh Quotient

Many embedding algorithms rely on eigenvalue decomposition: PCA, KPCA, LDA, Laplacian Eigenmaps.
These eigenvalue problems are all related to the Rayleigh Quotient.

Definition: Rayleigh Quotient

The Rayleigh Quotient for a fixed symmetric matrix $\boldsymbol{A} \in S^n\ (\Re)$ is a multivariate function $f: \Re^n - \{\mathbf{0}\} \rightarrow \Re$ defined by:

$$f(\boldsymbol{x}) = \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

Definition: Generalized Rayleigh Quotient

For a fixed symmetric matrix $\boldsymbol{A} \in S^n\ (\Re)$ and a positive definite matrix $\boldsymbol{B}$ of the same size, a generalized Rayleigh Quotient corresponding to them is given by $f: \Re^n - \{\mathbf{0}\} \rightarrow \Re$ defined by:

$$f(\boldsymbol{x}) = \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{B} \boldsymbol{x}}$$

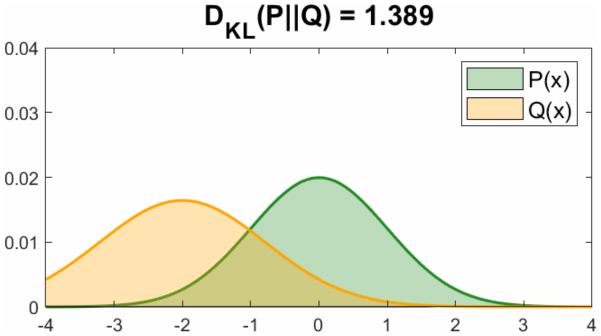
Solving eigenvalue problems are a lot faster and more reliable than doing gradient descent. This is why many dimensionality reduction methods are posed as Rayleigh quotient optimization problems.

PCA	$\max\limits_{\boldsymbol{\alpha} \neq \mathbf{0}} \dfrac{\boldsymbol{\alpha}^T \boldsymbol{\Sigma} \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \boldsymbol{\alpha}}$	$\boldsymbol{A} \leftarrow \boldsymbol{\Sigma}$ (Sample covariance)	$\boxed{\boldsymbol{\Sigma}\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha}}$
LDA	$\max\limits_{\boldsymbol{w} \neq \mathbf{0}} \dfrac{\boldsymbol{w}^T \boldsymbol{S}_b \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_w \boldsymbol{w}}$	$\boldsymbol{A} \leftarrow \boldsymbol{S}_b$ (Between-class scatter) $\boldsymbol{B} \leftarrow \boldsymbol{S}_w$ (Within-class scatter)	$\boxed{\boldsymbol{S}_b \boldsymbol{w} = \lambda \boldsymbol{S}_w \boldsymbol{w}}$
Laplacian Eigenmaps	$\min\limits_{\substack{\mathbf{v} \neq \mathbf{0} \\ \mathbf{v}^T \boldsymbol{D} \mathbf{1} = \mathbf{0}}} \dfrac{\mathbf{v}^T \boldsymbol{L} \mathbf{v}}{\mathbf{v}^T \boldsymbol{D} \mathbf{v}}$	$\boldsymbol{A} \leftarrow \boldsymbol{L}$ (Graph Laplacian) $\boldsymbol{B} \leftarrow \boldsymbol{D}$ (Degree matrix)	$\boxed{\boldsymbol{L}\mathbf{v} = \lambda \boldsymbol{D}\mathbf{v}}$

Source: https://www.sjsu.edu/faculty/guangliang.chen/Math250/lec4RQ.pdf

T-SNE

Van der Maaten (2008)



Definition: **Kullback-Leibler (KL) Divergence**

(Relative Entropy)

For *discrete* probability distributions P and Q, defined on the sample space χ, the KL divergence from Q to P is defined to be:

$$D_{KL}(P\,||\,Q) = \sum_{x\in\chi} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

For *continuous* probability distributions P and Q, the KL divergence from Q to P is defined to be:

$$D_{KL}(P\,||\,Q) = \int P(x)\log\left(\frac{P(x)}{Q(x)}\right)dx$$

Notes:

- $D_{KL}(P\,||\,Q) \neq D_{KL}(Q\,||\,P)$
- D_{KL} is always non-negative. $D_{KL} = 0$ iff $P(x) = Q(x)$

Step 1. Compute a probability-based similarity, p_{ij}, between data points, $\boldsymbol{x}$.

$$p_{j|i} = \frac{\exp\left(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2/2\sigma_i^2\right)}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \qquad p_{i|i} = 0$$

t-SNE performs a binary search for σ_i that produces a user-defined value of the **perplexity** $Perp(P_i) = 2^{H(P_i)}$, where $H(P_i)$ is the Shannon entropy:

$$H(P_i) = -\sum_j p_{j|i}\log_2 p_{j|i}$$

Step 2. Define a measure of similarity for $\boldsymbol{y}$ but using a Student's t-distribution with 1 DOF.

$$q_{ij} = \frac{\left(1 + \left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|\right)^{-1}}{\sum_{k\neq l}(1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|)^{-1}} \qquad q_{i|i} = 0$$

Step 3. Find $\boldsymbol{y}$ that minimize the KL divergence between P and Q via gradient descent.

$$D_{KL}(P\,||\,Q) = \sum_{\boldsymbol{i\neq j}} p_{ij}\log\left(\frac{p_{ij}}{q_{ij}}\right)$$

Notes:

- Since the t-SNE cost function is highly non-convex, optimality of $\boldsymbol{y}$ is not guaranteed and its results can vary on each run.
- t-SNE may be less successful if the intrinsic dimensionality of the data is still high. T-SNE is very useful for reduction to 2-D or 3-D or for visualization purposes only.
- Recommended value for Perplexity is from 5 to 50.

Side Note: Out-of-Sample Extensions

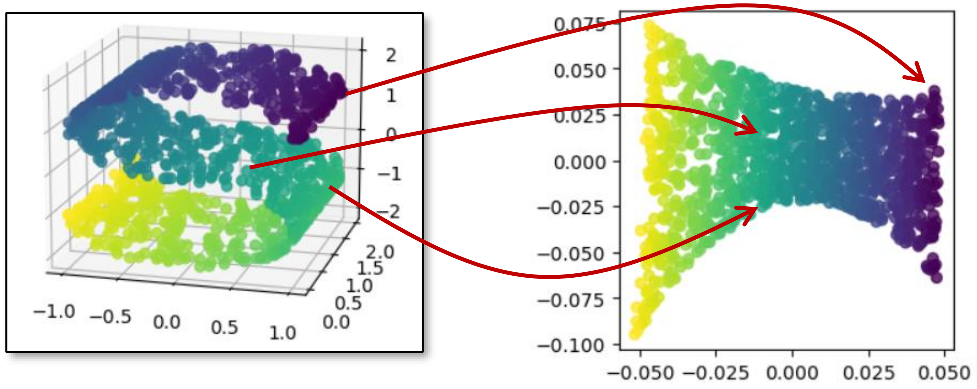
Bengio et al. (2003), van der Maaten (2009)

In their original versions, most manifold learning algorithms cannot produce an explicit *mapping function* from $\boldsymbol{x}$ to $\boldsymbol{y}$.

	Mapping Function
PCA	$\boldsymbol{y} = \boldsymbol{P}\boldsymbol{x}$
LDA	$\boldsymbol{y} = \boldsymbol{W}^T\boldsymbol{x}$
ICA	$\boldsymbol{S} = \boldsymbol{W}\boldsymbol{X}$
Kernel PCA	$\boldsymbol{y} = \boldsymbol{P}^T\boldsymbol{k}(\boldsymbol{x}, \boldsymbol{X})$
Isomap	????
t-SNE	????

Solution 1: Learn the mapping function using regression.

$\boldsymbol{x} \in \Re^3$



$\boldsymbol{y} \in \Re^2$

e.g. t-SNE becomes Parametric t-SNE

See van der Maaten (2009). Learning a Parametric Embedding by Preserving Local Structure. AISTATS.

Solution 2: Use the same manifold learning concept but enforce the mapping function in the derivation.

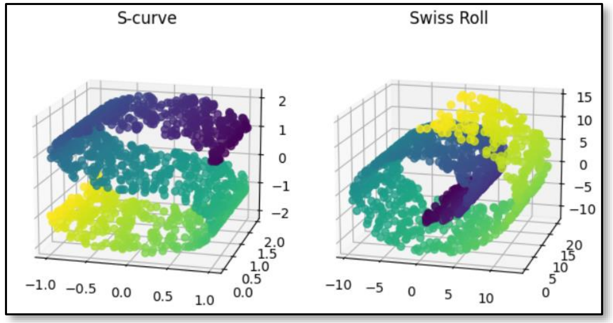
e.g. Laplacian Eigenmap becomes Locality Preserving Projections (LPP)

See He and Niyogi (2003). Locality Preserving Projections. NIPS 2003.

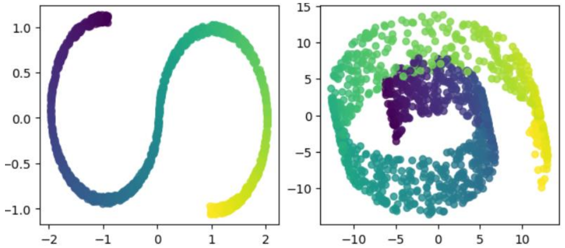
$$\boldsymbol{L}\mathbf{v} = \lambda\boldsymbol{D}\mathbf{v} \xrightarrow{\boldsymbol{y} = \boldsymbol{X}\mathbf{v}} \boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^T\mathbf{v} = \lambda\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^T\mathbf{v}$$

Example: S-curve and Swiss Roll Toy Data Sets

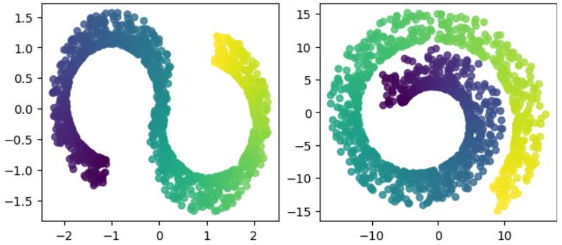
Perform various manifold learning algorithms on these 2 data sets, then compare the results.



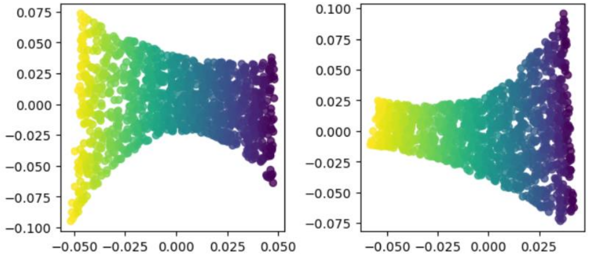
Principal Components Analysis (PCA)



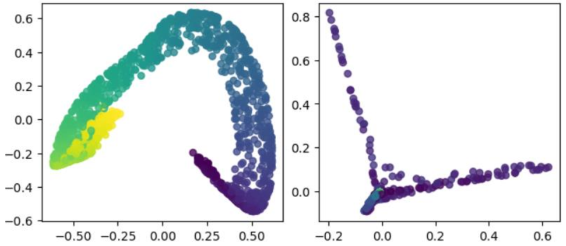
Multidimensional Scaling (MDS)



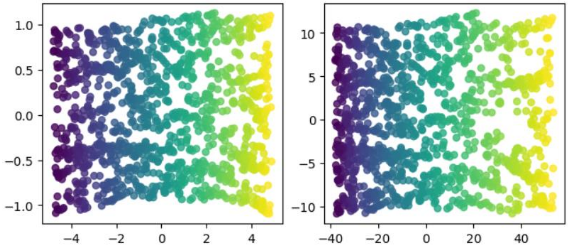
Locally Linear Embedding (LLE)



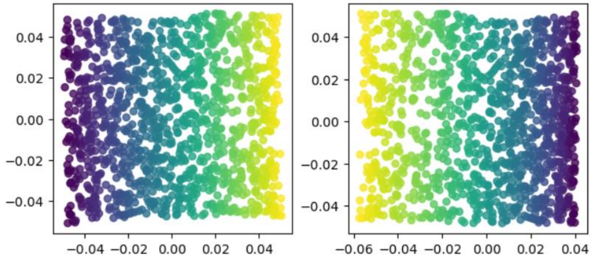
Kernel Principal Components Analysis (KPCA)



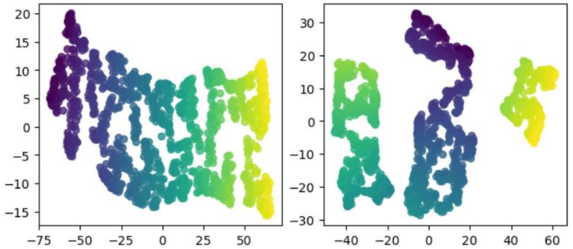
Isomap



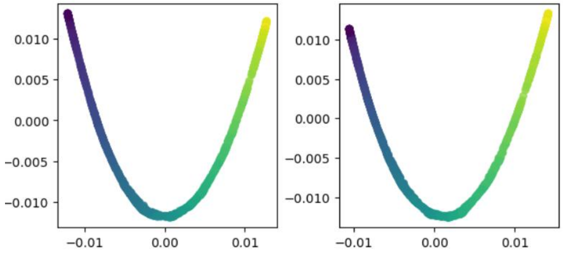
Hessian LLE



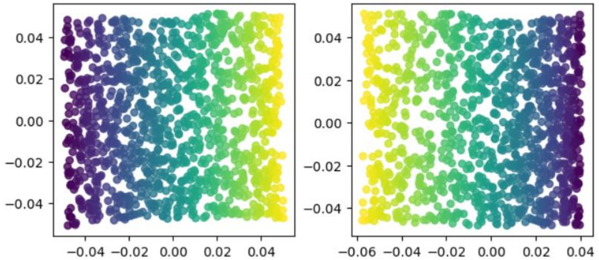
t-SNE



Laplacian Eigenmaps



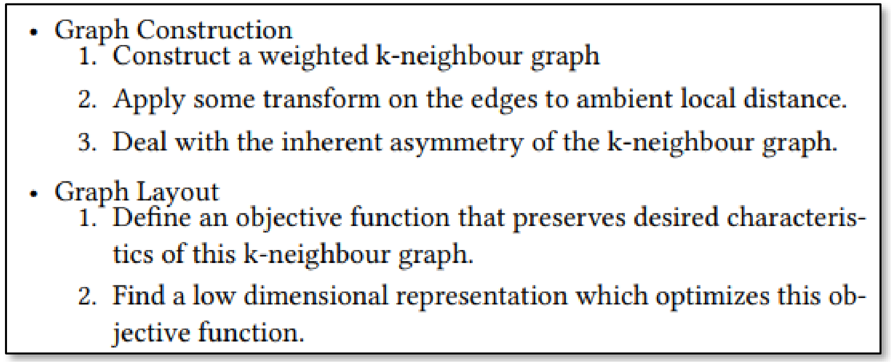
Local Tangent Space Alignment



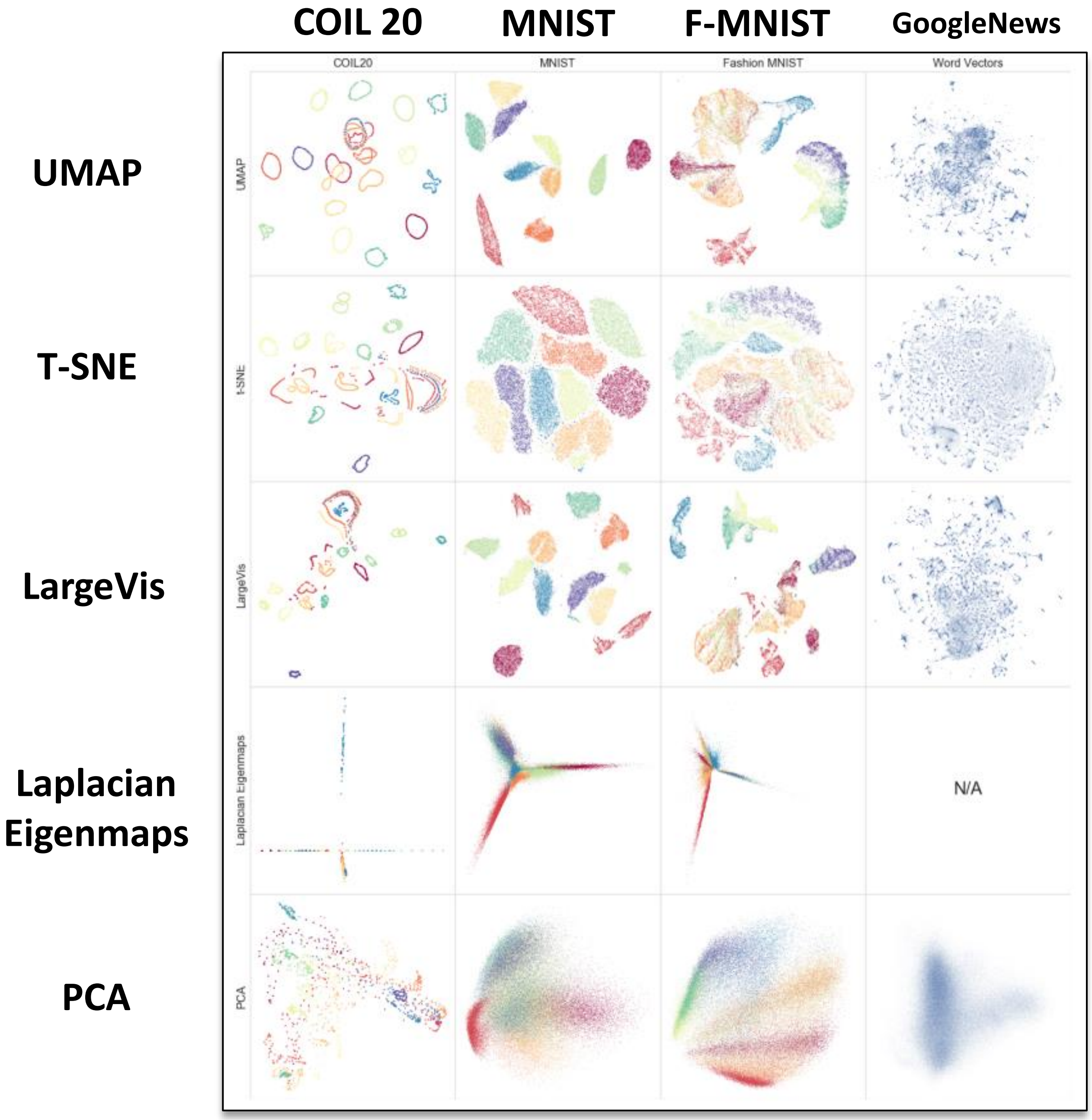
UMAP

Uniform Manifold Approximation and Projection
McInnes et al. (2020)

- Theoretical framework: Riemannian geometry and topology
- Competitive results with t-SNE
- More scalable to large data sets than t-SNE
- The algorithm is similar with most other dimensionality reduction algorithms:



- Graph Construction
  1. Construct a weighted k-neighbour graph
  2. Apply some transform on the edges to ambient local distance.
  3. Deal with the inherent asymmetry of the k-neighbour graph.
- Graph Layout
  1. Define an objective function that preserves desired characteristics of this k-neighbour graph.
  2. Find a low dimensional representation which optimizes this objective function.



COIL 20 MNIST F-MNIST GoogleNews

UMAP

T-SNE

LargeVis

Laplacian Eigenmaps

PCA

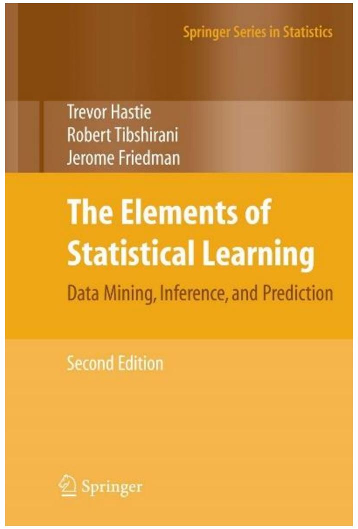
Outline

- Kernel Methods
  - Kernel PCA

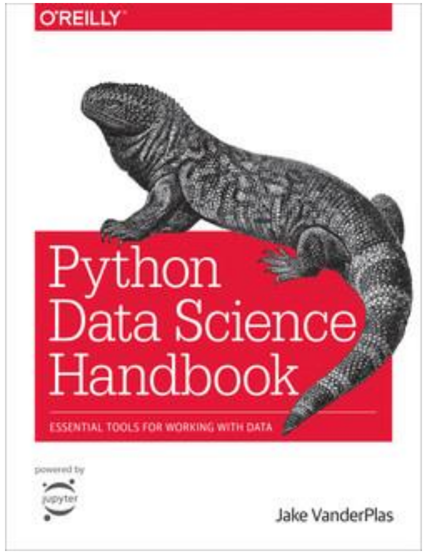
- Manifold Learning
  - Multidimensional Scaling
  - Isomap Embedding
  - Locally Linear Embedding (LLE)
  - Laplacian Eigenmaps
  - t-Distributed Stochastic Neighborhood Embedding (t-SNE)
  - Uniform Manifold Approximation and Projection (UMAP)

- Other Related Methods
  - *Sammon Mapping*
  - *Diffusion Maps*
  - *Autoencoders*
  - *LargeVis* https://arxiv.org/pdf/1602.00370.pdf

Hastie *et al.* (2008)
The Elements of Statistical Learning.
2nd Ed. Springer.


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

Jake VanderPlas (2016)
Python Data Science Handbook.
O'Reilly Media, Inc.


O'REILLY
Python Data Science Handbook
ESSENTIAL TOOLS FOR WORKING WITH DATA
powered by jupyter
Jake VanderPlas

Further Reading

- Bakır, Gökhan H., Jason Weston, and Bernhard Schölkopf. “Learning to find pre-images.” Advances in neural information processing systems 16 (2004): 449-456.

- KPCA paper: Scholkopf et al. (1998). “Nonlinear Component Analysis as a Kernel Eigenvalue Problem.” Neural Computation, 10, 1299-1319. https://www.mlpack.org/papers/kpca.pdf

- Anowar et al. (2021). Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE). Computer Science Review. https://www.sciencedirect.com/science/article/abs/pii/S1574013721000186

- L.J.P. van der Maaten, E.O. Postma, and H.J. van den Herik. Dimensionality Reduction: A Comparative Review. Tilburg University Technical Report, TiCC-TR 2009-005, 2009. http://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf

- Pilario et al. (2020). “A Review of Kernel Methods for Feature Extraction in Nonlinear Process Monitoring.” MDPI Processes, 8(1), no. 24. https://www.mdpi.com/2227-9717/8/1/24

- MDS paper: Kruskal (1964). “Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis.” Psychometrika, 29. http://cda.psych.uiuc.edu/psychometrika_highly_cited_articles/kruskal_1964a.pdf

- Isomap paper: Tenenbaum et al. (2000). “A global geometric framework for nonlinear dimensionality reduction.” Science 290 (5500). https://www.science.org/doi/10.1126/science.290.5500.2319

- LLE paper: Roweis et al. (2000). “Nonlinear dimensionality reduction by locally linear embedding.” Science 290:2323 (2000). https://www.science.org/doi/10.1126/science.290.5500.2323

- Saul, L.K.; Roweis, S. Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds. J. Mach. Learn. Res. 2003, 4, 119–155.

- Hessian Eigenmaps paper: Donoho et al. (2003). “Hessian Eigenmaps: Locally linear embedding techniques for high-dimensional data”. PNAS 100:5591. https://www.pnas.org/doi/10.1073/pnas.1031596100

- Laplacian Eigenmaps paper: Belkin and Niyogi (2003). “Laplacian Eigenmaps for Dimensionality Reduction and Data Representation.” Neural Computation, June 2003; 15 (6):1373-1396. https://web.cse.ohio-state.edu/~belkin.8/papers/LEM_NC_03.pdf

- T-SNE paper: van der Maaten et al. (2008). “Visualizing High-Dimensional Data Using t-SNE.” Journal of Machine Learning Research. 9(86):2579−2605, 2008. https://jmlr.org/papers/v9/vandermaaten08a.html

Further Reading

- LTSA paper: Zhang and Zha (2002). “Principal Manifolds And Nonlinear Dimension Reduction via Local Tangent Space Alignment.” https://arxiv.org/pdf/cs/0212008.pdf
- Sammon mapping paper: Sammon J.W. (1969). "A nonlinear mapping for data structure analysis". IEEE Transactions on Computers. 18 (5): 401–409. doi: 10.1109/t-c.1969.222678. http://theoval.cmp.uea.ac.uk/~gcc/matlab/sammon/sammon.pdf
- LPP paper: He and Niyogi (2003). “Locality Preserving Projections.” https://proceedings.neurips.cc/paper/2003/file/d69116f8b0140cdeb1f99a4d5096ffe4-Paper.pdf
- https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html
- https://scikit-learn.org/stable/modules/manifold.html
- Bengio et al. (2003). “Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering.” Advances in Neural Information Processing Systems. https://proceedings.neurips.cc/paper/2003/file/cf05968255451bdefe3c5bc64d550517-Paper.pdf
- MATLAB Toolbox for Dimensionality Reduction (by van der Maaten): http://lvdmaaten.github.io/drtoolbox/
- UMAP paper: https://arxiv.org/pdf/1802.03426.pdf
  - Datasets used in the slide:
  - **COIL20:** Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (coil-20. Technical report, 1996
  - **MNIST**: Yann Lecun and Corinna Cortes. ŒMNIST database of handwritten digits.
  - **Fashion MNIST**: Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. CoRR, abs/1708.07747, 2017.
  - **GoogleNews**: Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems, 3111–3119, 2013.

Proof

In the derivation of Laplacian Eigenmap, let's prove that:

$$\boxed{\frac{1}{2}\sum_{ij} w_{ij}\left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^2 = \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}}$$

where: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \in \Re^{n\times n}$

$\boldsymbol{D}$ = degree matrix

$\boldsymbol{W}$ = *weights matrix* or *adjacency matrix* if graph is unweighted.

Expanding the left-hand side:

$$\frac{1}{2}\sum_{ij} w_{ij}\left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^2 = \frac{1}{2}\left[\sum_{i,j} w_{ij}\boldsymbol{y}_i^2 + \sum_{i,j} w_{ij}\boldsymbol{y}_j^2 - 2\sum_{i,j} w_{ij}\boldsymbol{y}_i\boldsymbol{y}_j\right]$$

Note that $\boldsymbol{W}\mathbf{1} = \boldsymbol{D}$, where $\mathbf{1} = [1,1,1,1,\ldots]^T$ which means that the sum of the columns (or rows) of the weights matrix gives the degree matrix.

Hence, $\sum_i w_{ij} = d_i$ and $\sum_j w_{ij} = d_j$.

$$\frac{1}{2}\sum_{ij} w_{ij}\left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^2 = \frac{1}{2}\left[\sum_{i} d_i\boldsymbol{y}_i^2 + \sum_{j} d_j\boldsymbol{y}_j^2 - 2\sum_{i,j} w_{ij}\boldsymbol{y}_i\boldsymbol{y}_j\right]$$

Since $\boldsymbol{D}$ is a diagonal matrix, then $d_i = d_j$:

$$\frac{1}{2}\sum_{ij} w_{ij}\left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^2 = \frac{1}{2}\left[2\sum_{j} d_j\boldsymbol{y}_j^2 - 2\sum_{i,j} w_{ij}\boldsymbol{y}_i\boldsymbol{y}_j\right]$$

Bringing them back into matrix notation:

$$\frac{1}{2}\left[2\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y} - \mathbf{2}\boldsymbol{y}^T\boldsymbol{W}\boldsymbol{y}\right]$$

$$= \boldsymbol{y}^T(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{y}$$

$$= \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y} \quad \text{Q.E.D.}$$

Team Project Guidelines

1. Form 1-3 members per team.

2. Find a problem + data set that requires an ML solution.
   - Solve the problem using the ML methods discussed in class.
   - Present your results face-to-face.
   - NO two teams should have the same problem.

Grading: Oral Presentation (40%)
Written Report (60%)

NOTE: Your workflow must have:

- 1 pre-processing step
- 1 unsupervised learning step
- 1 supervised learning step