



# Nonlinear Dimensionality Reduction: Kernel PCA and Manifold Learning

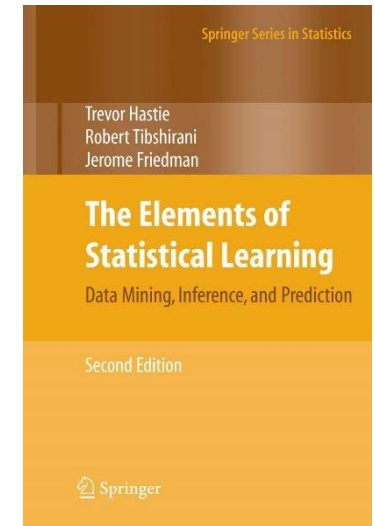
**Assoc. Prof. Karl Ezra Pilario, Ph.D.**

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

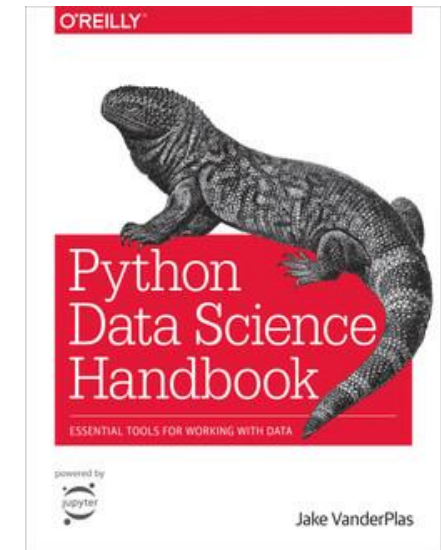
# Outline

- Kernel Methods
  - Kernel PCA
- Manifold Learning
  - Multidimensional Scaling
  - Isomap Embedding
  - Locally Linear Embedding (LLE)
  - Laplacian Eigenmaps
  - t-Distributed Stochastic Neighborhood Embedding (t-SNE)

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



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



# 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  $\mathbf{X}$ :  $\Sigma = \frac{1}{N-1} \mathbf{X}^T \mathbf{X}$

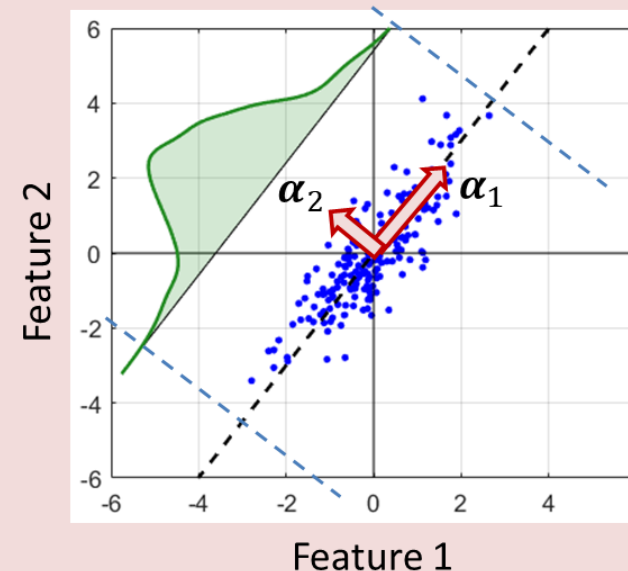
3. Compute the eigenvalue decomposition of  $\Sigma$ :  $\Sigma = \mathbf{V} \Lambda \mathbf{V}^T$

4. Choose only  $n$  principal components, then get  $\mathbf{Y}$ :  
 $\mathbf{P} = \mathbf{V}_n$   
 $\mathbf{Y} = \mathbf{X}\mathbf{P}$

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

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

where:  $\mathbf{Y} = [\mathbf{y}_1 \ \mathbf{y}_2 \ \dots \ \mathbf{y}_m]$   $\mathbf{y}$ 's are called **scores**.  
 $\mathbf{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_m]$   
 $\mathbf{P} = [\boldsymbol{\alpha}_1 \ \boldsymbol{\alpha}_2 \ \dots \ \boldsymbol{\alpha}_m]$   $\boldsymbol{\alpha}$ 's are called **loadings / coefficients**.



$(\mathbf{y}_1, \boldsymbol{\alpha}_1)$  is the **1st** principal component.  
 $(\mathbf{y}_2, \boldsymbol{\alpha}_2)$  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.

$\begin{matrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_m \end{matrix} \quad \Rightarrow \quad \begin{matrix} \mathbf{y}_1 \text{ (PC1)} \\ \mathbf{y}_2 \text{ (PC2)} \end{matrix}$

# 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  $\mathbf{X}$ :  $\mathbf{\Sigma} = \frac{1}{N-1} \mathbf{X}^T \mathbf{X}$

3. Compute the eigenvalue decomposition of  $\mathbf{\Sigma}$ :  $\mathbf{\Sigma} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$

4. Choose only  $n$  principal components, then get  $\mathbf{Y}$ :  
 $\mathbf{P} = \mathbf{V}_n$   
 $\mathbf{Y} = \mathbf{X} \mathbf{P}$

## 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,  $\mathbf{K}$ , using the kernel  $k$ :  $\mathbf{K} = [k(\mathbf{x}_i, \mathbf{x}_j)]$

4. Center the kernel matrix:  $\mathbf{K}_c = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N$

5. Compute the eigenvalue decomposition of  $\mathbf{K}_c$ :  $\mathbf{K}_c = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$

6. Choose only  $n$  principal components:  $\mathbf{P} = \mathbf{\Lambda}^{-1/2} \mathbf{V}_n$   
 $\mathbf{Y} = \mathbf{P}^T \mathbf{K}_c$

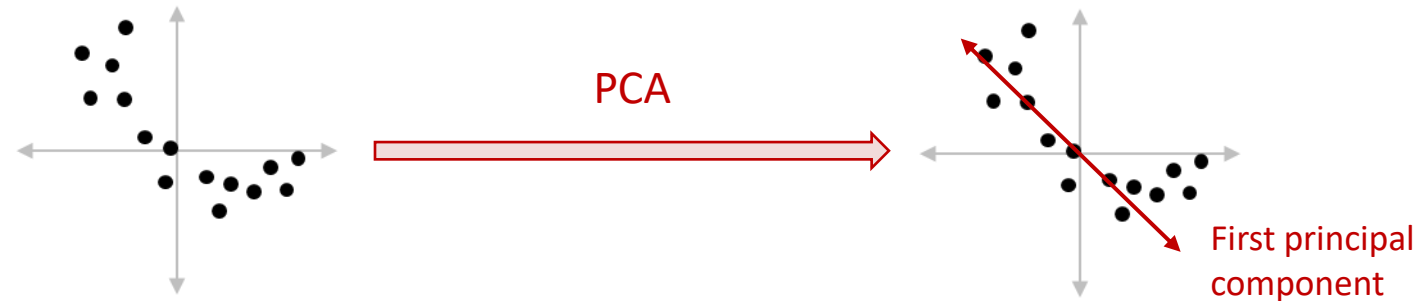
To project any new test sample,  $\mathbf{x}'$ :  $\mathbf{y} = \mathbf{P}^T \mathbf{k}(\mathbf{x}', \mathbf{X})$

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

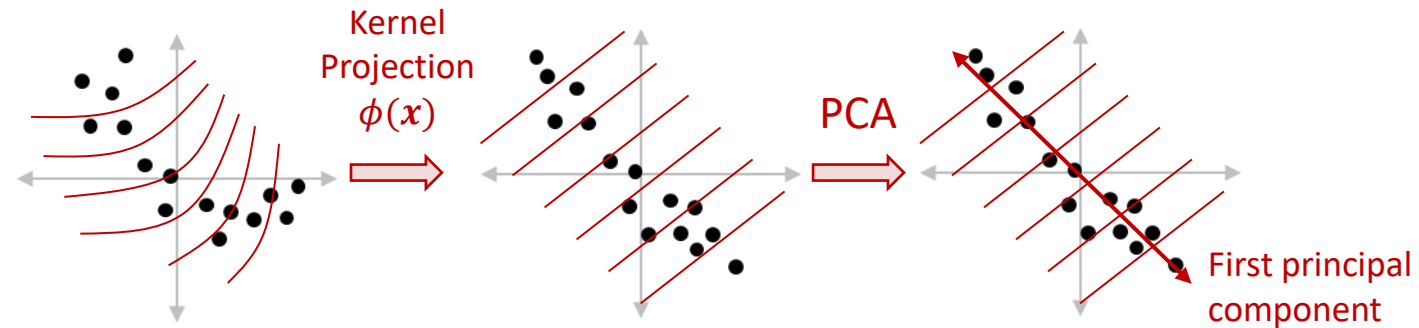
# Kernel PCA

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



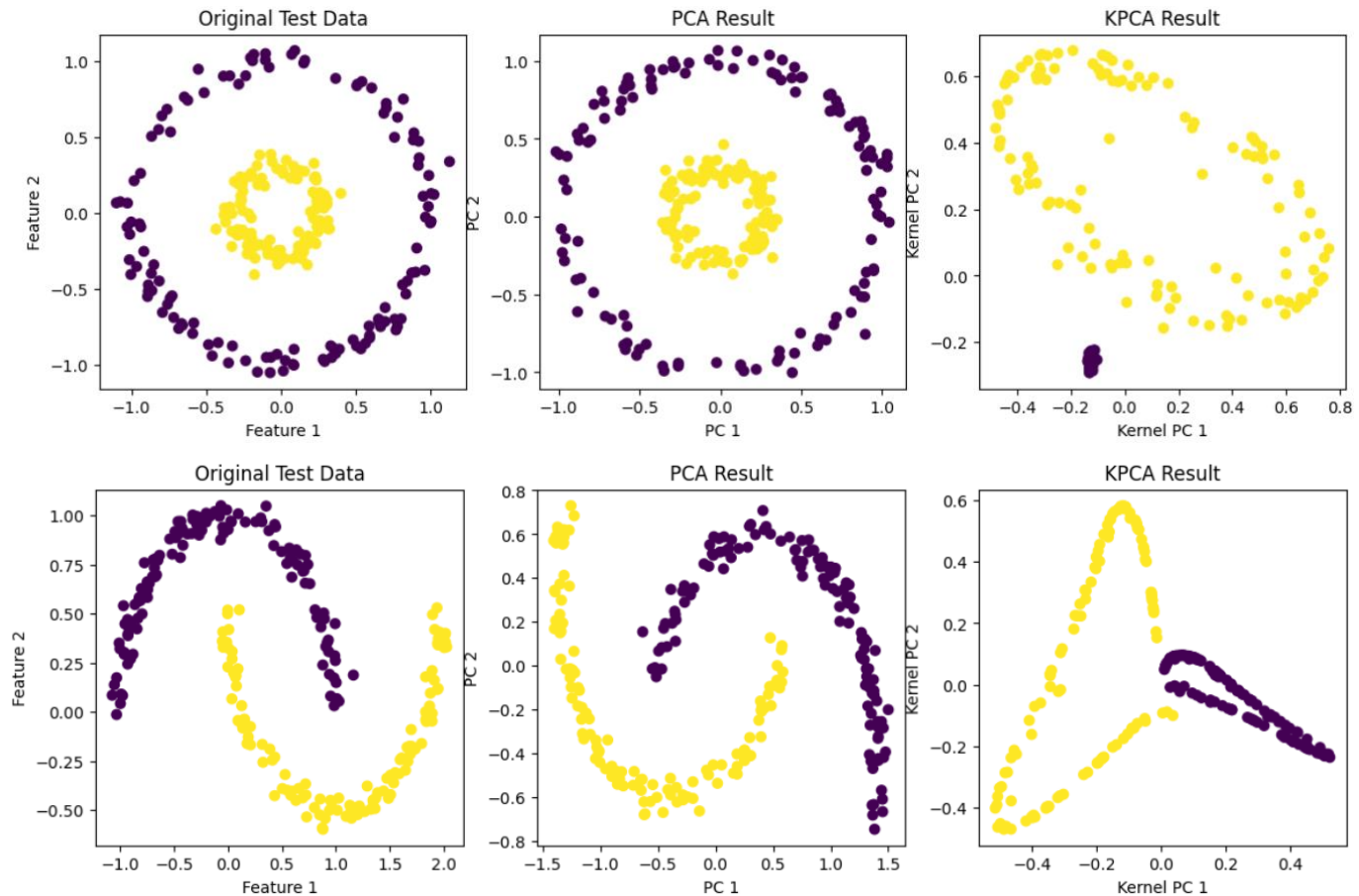
- The **PCA covariance** is computed as:  $\Sigma = \frac{1}{N-1} \mathbf{X}\mathbf{X}^T$
- The **KPCA covariance** is approximated by the kernel matrix:  $\Sigma = \frac{1}{N-1} \phi(\mathbf{X})\phi(\mathbf{X})^T = \mathbf{K} = [k(x_i, x_j)]$

# 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

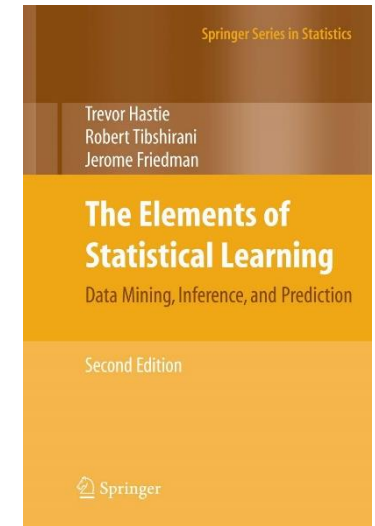
Generate 1000 samples of the concentric circles data (make\_circles) and two moons data (make\_moons). Split them into 75% training and 25% testing. Apply PCA and KPCA (RBF, gamma=10) on the training data, then transform the test data. Compare the results.



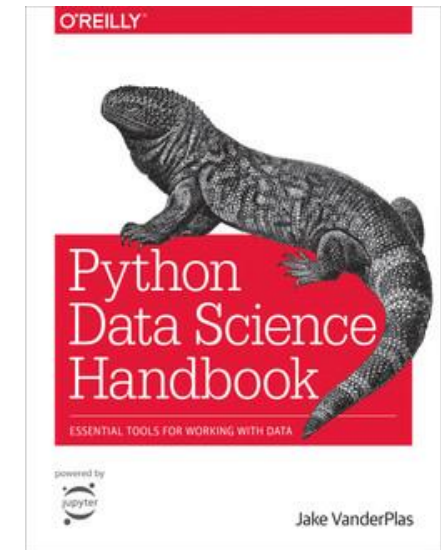
# Outline

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

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



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





# Manifold Learning

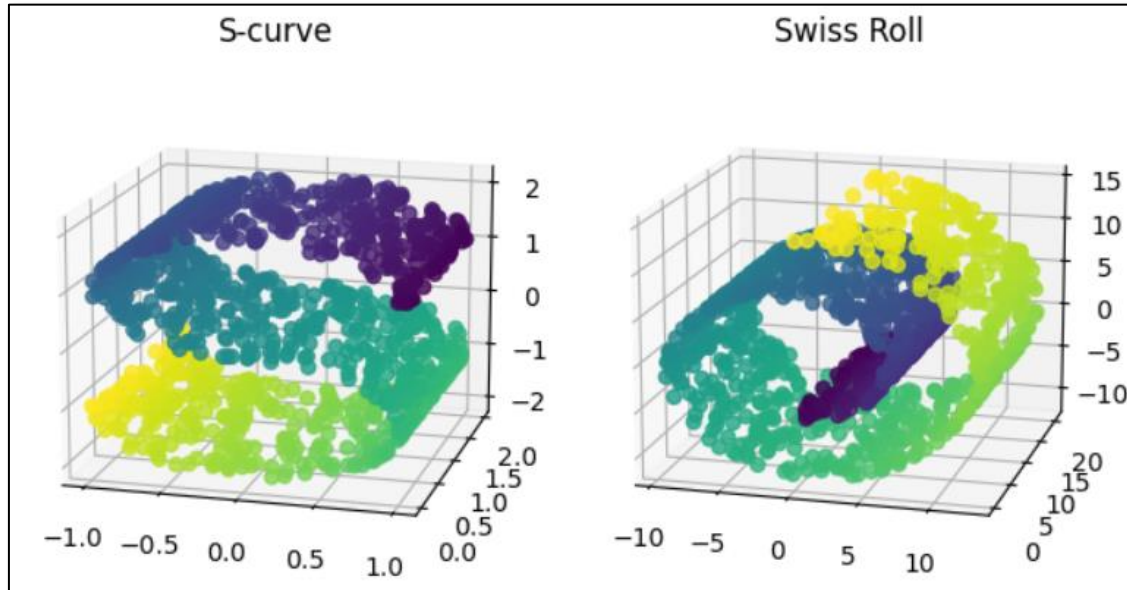
Manifold learning algorithms and KPCA are generalizations of PCA. They seek low-dimensional manifolds embedded in high-dimensional spaces.

(Python Data Science Handbook, Jake VanderPlas, 2016)

Notation:

$$x_i \in \mathbb{R}^m \xrightarrow{\text{embedding}} y_i \in \mathbb{R}^d, \quad d < m$$

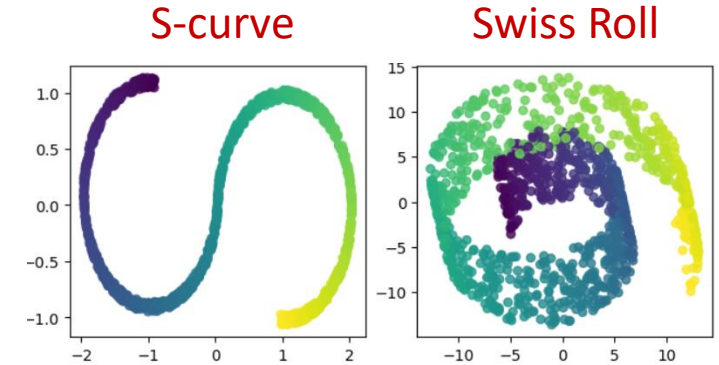
Latent Space



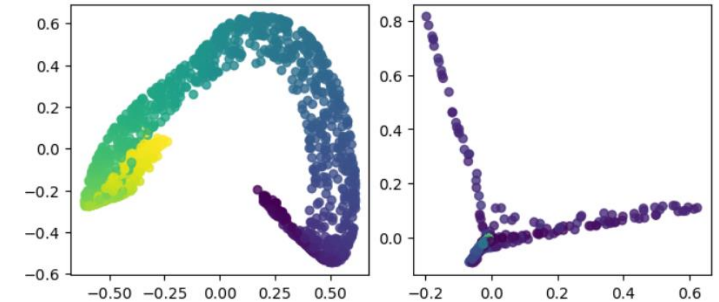
The **S-curve** and **Swiss Roll** data sets are two popular examples that illustrate the weakness of PCA and KPCA, and the strength of other manifold learning algorithms for nonlinear dimensionality reduction.

## 2-D Embedding of the Data Sets

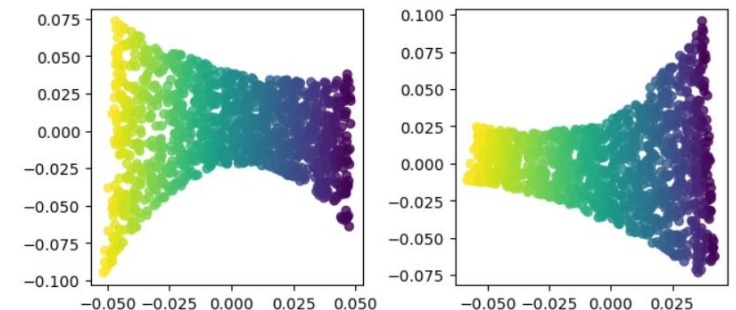
Principal  
Components  
Analysis (PCA)



Kernel Principal  
Components  
Analysis (KPCA)



Locally Linear  
Embedding  
(LLE)

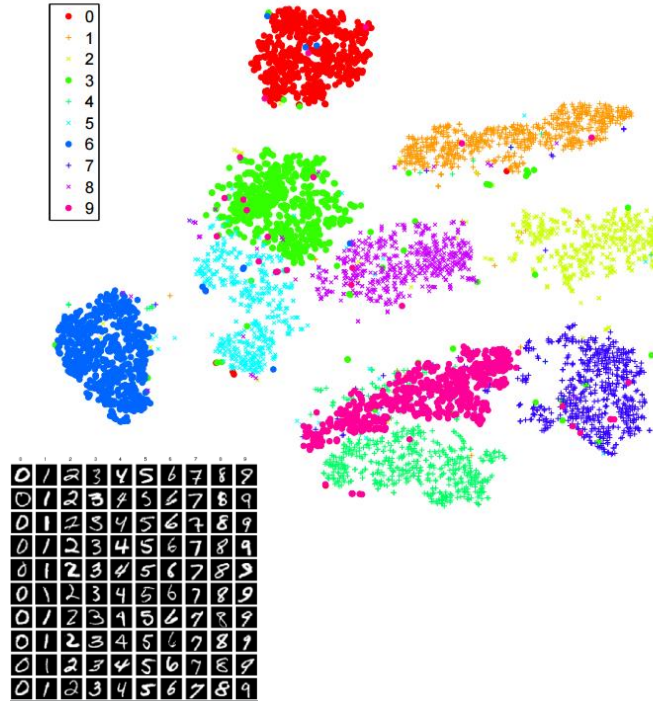
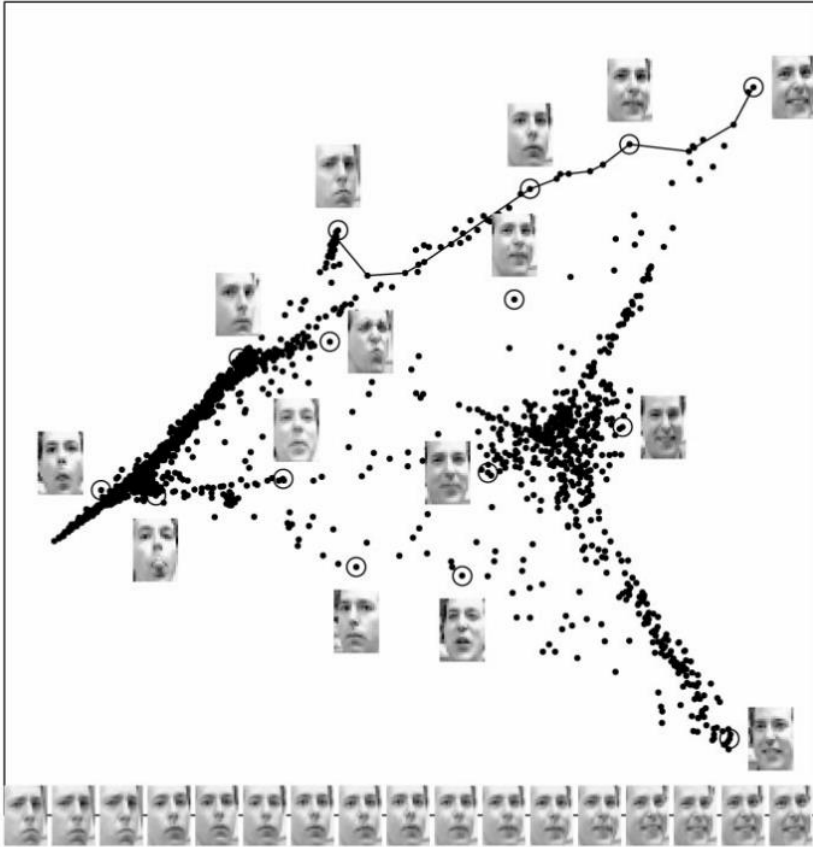




# Manifold Learning

Manifold learning algorithms have many applications.

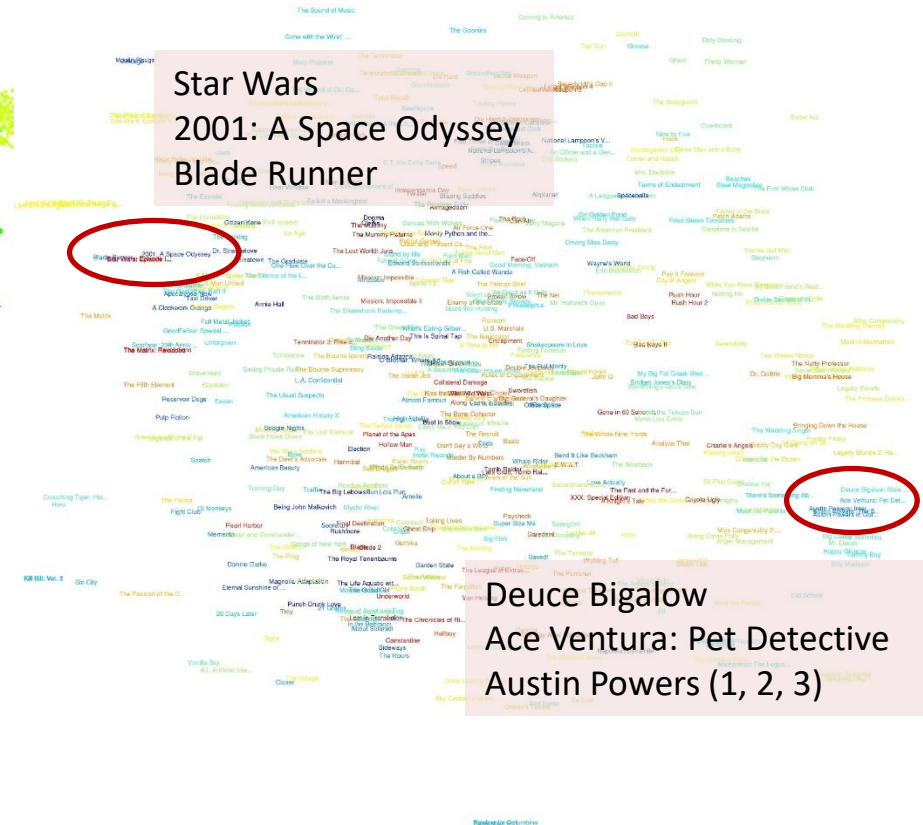
LLE applied to face images can reveal paths of varying pose and expression. (Roweis, 2000)



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](https://lvdmaaten.github.io/tsne/examples/netflix_tsne.jpg)



# 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.

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

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

(Dissimilarity matrix)

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

$$\min_{\mathbf{y}} \sum_{i < j} (\|\mathbf{y}_i - \mathbf{y}_j\| - d_{i,j})^2$$

## Isomap

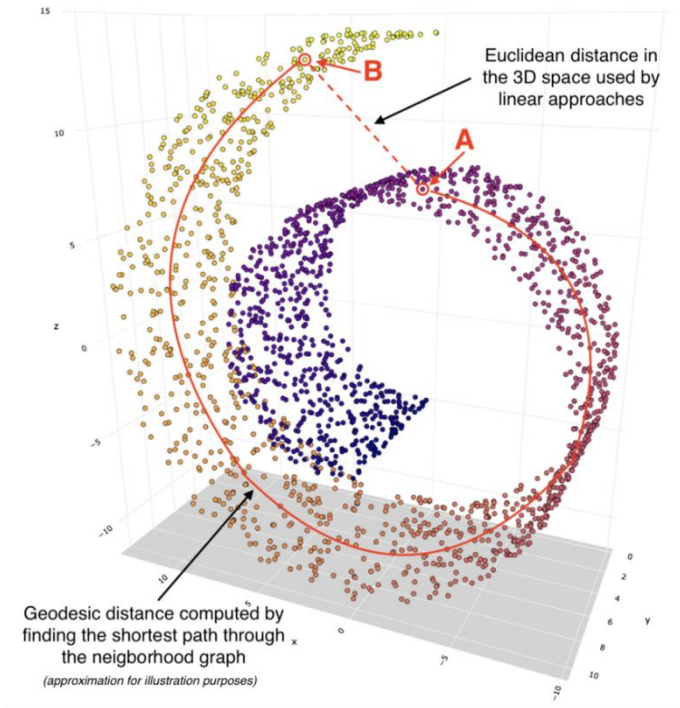
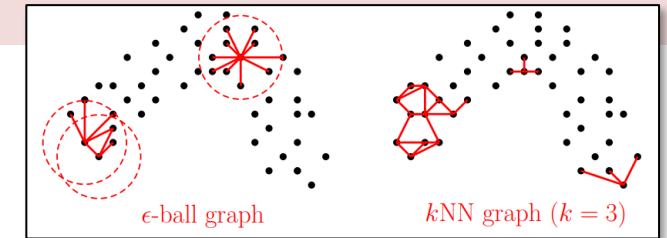
**Step 1.** Find all  $k$ -nearest neighbors of all samples.

**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  $\mathbf{y}_i$  such that the pairwise distances are *preserved as much as possible*. Use gradient descent.

$$\min_{\mathbf{y}} \sum_{i < j} (\|\mathbf{y}_i - \mathbf{y}_j\| - d_{i,j})^2$$



MDS → Euclidean distance  
Isomap → Geodesic distance

# 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.

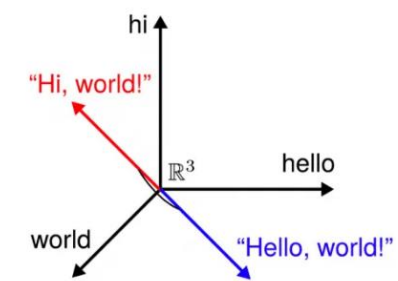
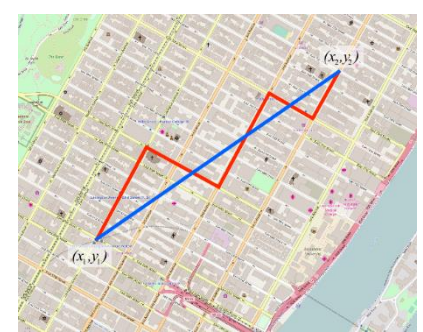
Hamming Distance,  
Edit Distance,  
Levenshtein Distance

ATCGGTAGT  
ATGGTTCCT

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	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



Euclidean

Cosine

Hamming

A	1	0	1	1	0	0
B	1	1	1	0	0	0

Manhattan

Minkowski

Chebyshev

Jaccard

Haversine

Sørensen-Dice

<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  $x_i$  from its neighbors.

$$\min_{w_{ij}} \sum_i \left[ x_i - \sum_j w_{ij} x_j \right]^2$$

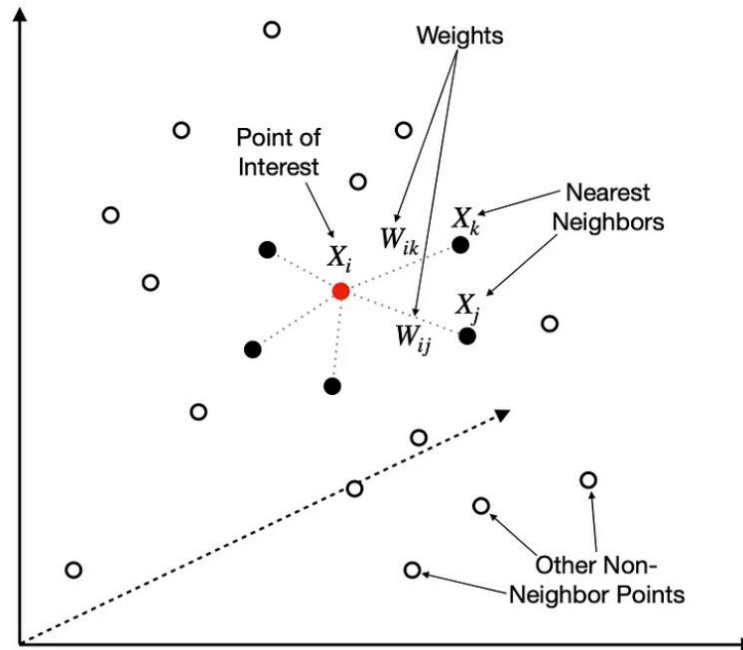
s.t.  $\sum_j w_{ij} = 1$  for neighbors of sample  $i$   
 $w_{ij} = 0$  for non-neighbors of sample  $i$

**Step 3.** Find vectors  $y_i$  that retain the reconstruction weights  $w_{ij}$ , but lie in low-dimensional space.

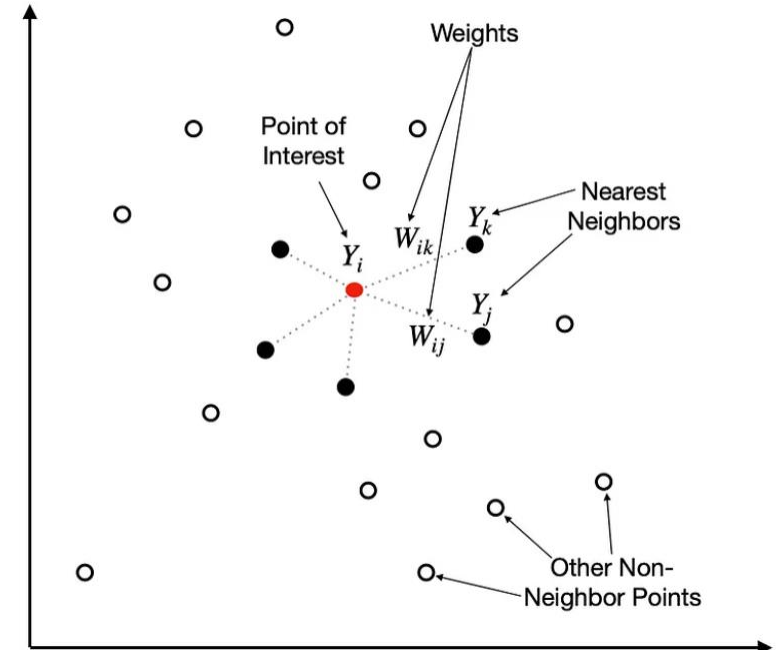
$$\min_y \sum_i \left[ y_i - \sum_j w_{ij} y_j \right]^2$$

**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

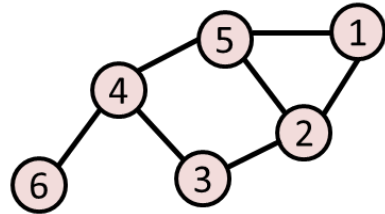
a.k.a. Spectral Embedding. *Belkin and Niyogi (2003)*

## Definition: Graph Laplacian

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

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \in \mathbb{R}^{n \times n}$$

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



**Example:**

$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$
Degree matrix	Adjacency matrix

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

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

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

### Interpretation:

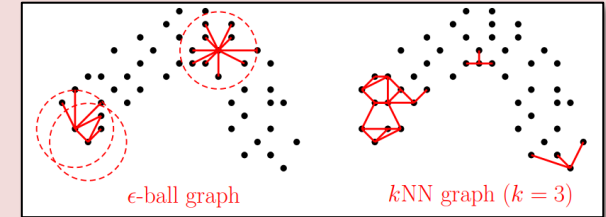
If  $\mathbf{x}_i$  and  $\mathbf{x}_j$  are close to each other in the original space, then  $w_{ij}$  (the  $i, j$ -th element of  $\mathbf{W}$ ) is close to 1 and the minimizer forces  $\mathbf{y}_i$  to be close to  $\mathbf{y}_j$  in the reduced space. Similarly, if  $\mathbf{x}_i$  and  $\mathbf{x}_j$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

### Where does the Graph Laplacian come in?

Well, it turns out that:

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

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



**Step 2.** Assign weights,  $\mathbf{W} \in \mathbb{R}^{n \times n}$ , to the graph edges:

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

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

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

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

- 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:

$$\mathbf{Y} = [\mathbf{v}_2, \mathbf{v}_3, \dots, \mathbf{v}_d] \in \mathbb{R}^{n \times d}$$

[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  $A \in S^n(\mathbb{R})$  is a multivariate function  $f: \mathbb{R}^n - \{0\} \rightarrow \mathbb{R}$  defined by:

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

### Definition: Generalized Rayleigh Quotient

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

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

### Theorem

If a given symmetric matrix  $A \in S^n(\mathbb{R})$  has  $\lambda_1$  and  $\lambda_n$  as its largest and smallest eigenvalues, with associated eigenvectors,  $v_1$  and  $v_n$ , then the **global maximum and minimum of its Rayleigh Quotient** are exactly  $\lambda_1$  and  $\lambda_n$ , respectively, and they occur at  $x = \pm v_1$  and  $x = \pm v_n$ , respectively.

$$\max_{\substack{x \neq 0 \\ x^T x = 1}} \frac{x^T A x}{x^T x} = \max_{\substack{x \neq 0 \\ x^T x = 1}} x^T A x = \lambda_1 @ x = \pm v_1$$

$$\min_{\substack{x \neq 0 \\ x^T x = 1}} \frac{x^T A x}{x^T x} = \min_{\substack{x \neq 0 \\ x^T x = 1}} x^T A x = \lambda_n @ x = \pm v_n$$

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

The Lagrangian can be written as:

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

Taking the partial derivative w.r.t.  $x$ :

$$\frac{\partial \mathcal{L}}{\partial x} = 2Ax - \lambda(2x) = 0 \quad \longrightarrow \quad \boxed{Ax = \lambda x}$$

**Definition of the Eigenvalue**

## 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  $A \in S^n(\mathbb{R})$  is a multivariate function  $f: \mathbb{R}^n - \{0\} \rightarrow \mathbb{R}$  defined by:

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

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

$$\max_{\alpha \neq 0} \frac{\alpha^T \Sigma \alpha}{\alpha^T \alpha}$$

$A \leftarrow \Sigma$  (Sample covariance)

$$\Sigma \alpha = \lambda \alpha$$

### Definition: Generalized Rayleigh Quotient

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

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

LDA

$$\max_{w \neq 0} \frac{w^T S_b w}{w^T S_w w}$$

$A \leftarrow S_b$  (Between-class scatter)  
 $B \leftarrow S_w$  (Within-class scatter)

$$S_b w = \lambda S_w w$$

Laplacian  
Eigenmaps

$$\min_{\substack{v \neq 0 \\ v^T D \mathbf{1} = 0}} \frac{v^T L v}{v^T D v}$$

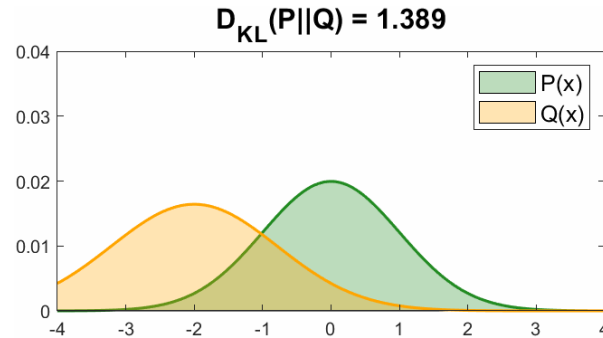
$A \leftarrow L$  (Graph Laplacian)  
 $B \leftarrow D$  (Degree matrix)

$$L v = \lambda D v$$



# 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  $\mathcal{X}$ , the KL divergence from  $Q$  to  $P$  is defined to be:

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

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,  $\mathbf{x}$ .

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

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

t-SNE performs a binary search for  $\sigma_i$  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  $\mathbf{y}$  but using a Student's t-distribution with 1 DOF.

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

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

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

Notes:

- Since the t-SNE cost function is highly non-convex, optimality of  $\mathbf{y}$  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  $x$  to  $y$ .

## Mapping Function

PCA

$$y = Px$$

LDA

$$y = W^T x$$

ICA

$$S = WX$$

Kernel PCA

$$y = P^T k(x, X)$$

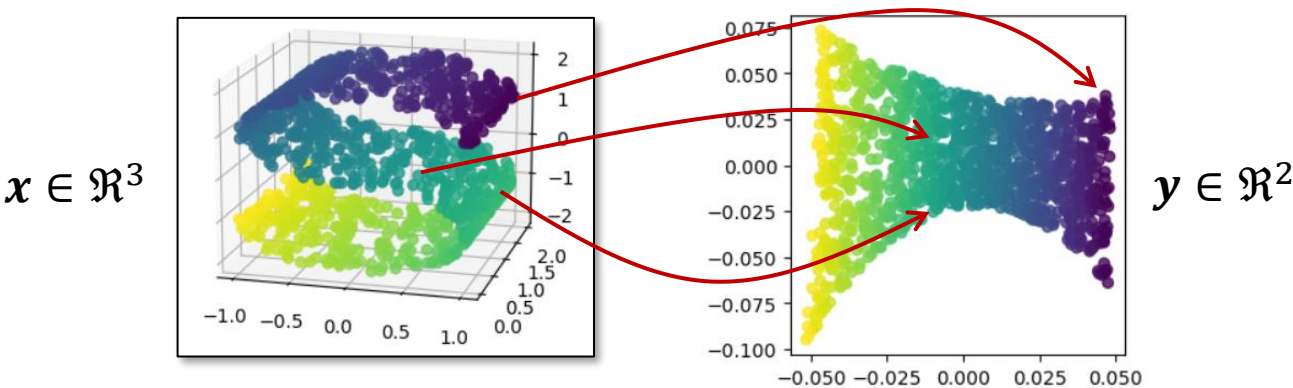
Isomap

????

t-SNE

????

**Solution 1:** Learn the mapping function using regression.



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.

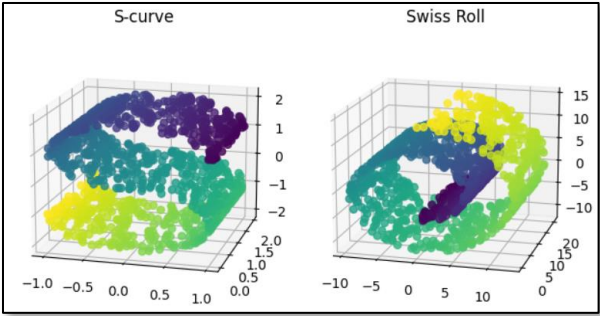
e.g. Laplacian Eigenmap becomes **Locality Preserving Projections (LPP)**

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

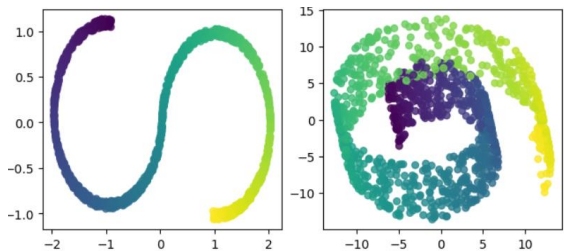
$$Lv = \lambda Dv \xrightarrow{y = Xv} XLX^T v = \lambda XDX^T v$$

# Example: S-curve and Swiss Roll Toy Data Sets

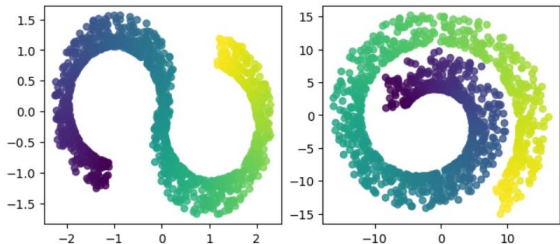
Perform various manifold learning algorithms on these 2 data sets, then compare the results.



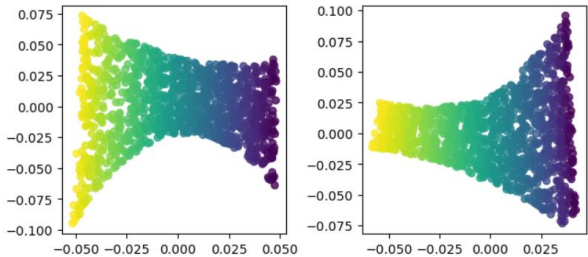
Principal Components Analysis (PCA)



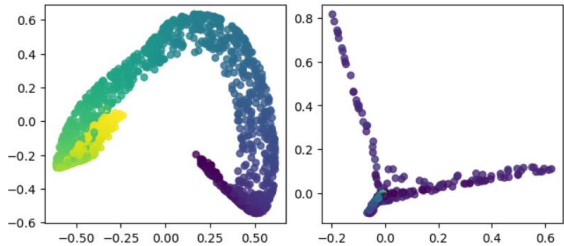
Multidimensional Scaling (MDS)



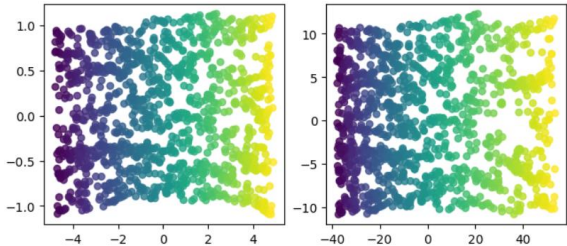
Locally Linear Embedding (LLE)



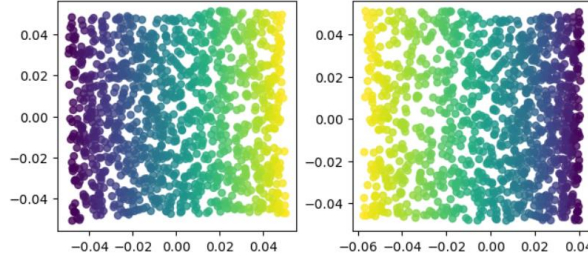
Kernel Principal Components Analysis (KPCA)



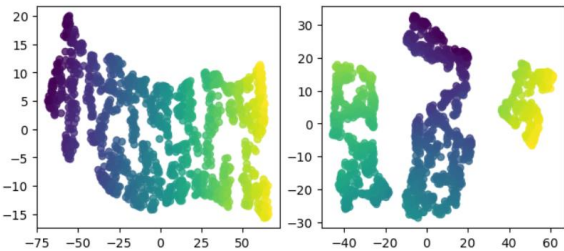
Isomap



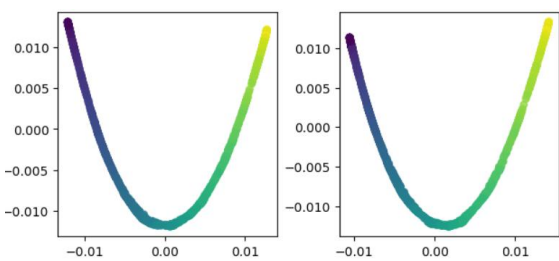
Hessian LLE



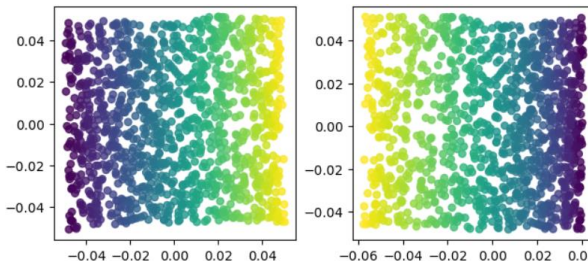
t-SNE



Laplacian Eigenmaps



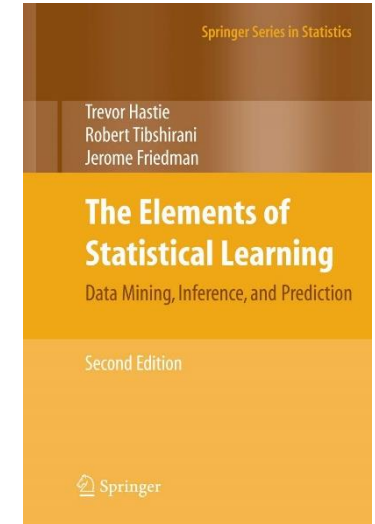
Local Tangent Space Alignment



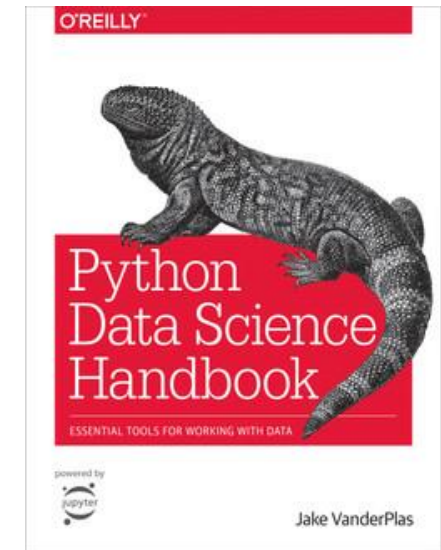
# Outline

- Kernel Methods
  - Kernel PCA
- Manifold Learning
  - Multidimensional Scaling
  - Isomap Embedding
  - Locally Linear Embedding (LLE)
  - Laplacian Eigenmaps
  - t-Distributed Stochastic Neighborhood Embedding (t-SNE)
- Other Related Methods
  - Sammon Mapping
  - Diffusion Maps
  - UMAP (Uniform Manifold Approximation and Projection)  
<https://arxiv.org/pdf/1802.03426.pdf>
  - LargeVis <https://arxiv.org/pdf/1602.00370.pdf>

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



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



# 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](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](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](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/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/>



# Proof

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

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

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

$\mathbf{D}$  = degree matrix

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

Expanding the left-hand side:

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

Note that  $\mathbf{W}\mathbf{1} = \mathbf{D}$ , where  $\mathbf{1} = [1, 1, 1, 1, \dots]^T$  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  $\sum_j w_{ij} = d_j$ .

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

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

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

Bringing them back into matrix notation:

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

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

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