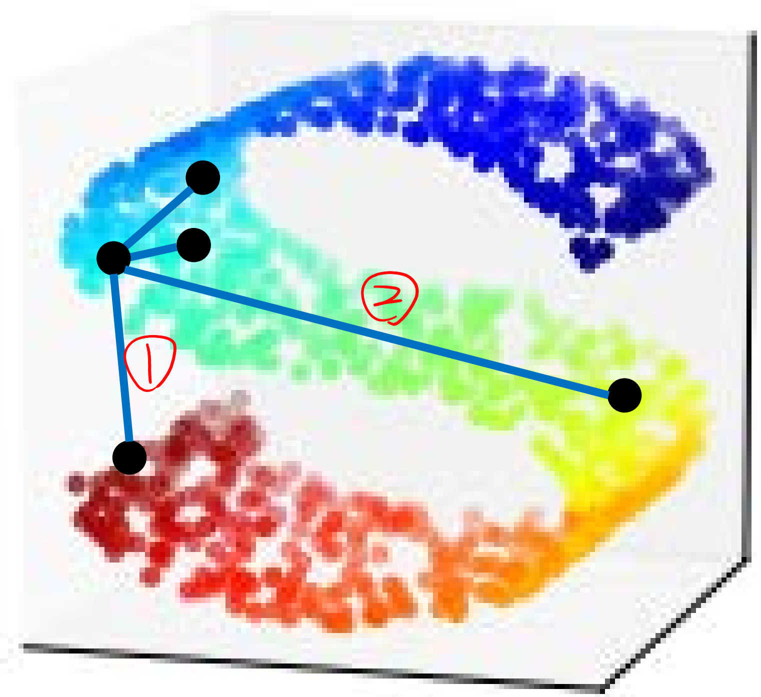
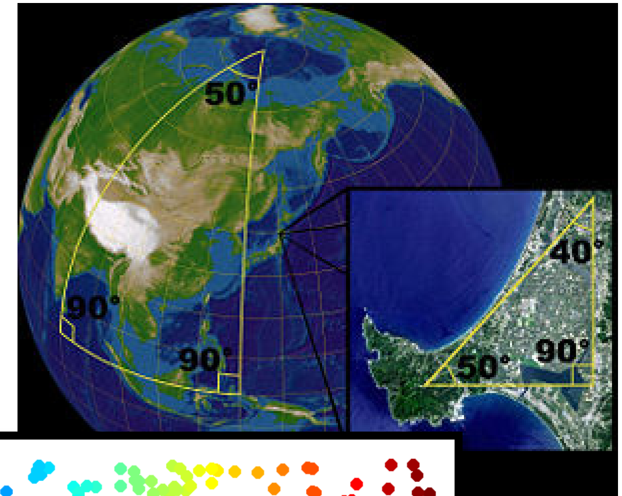
Unsupervised Learning: Neighbor Embedding

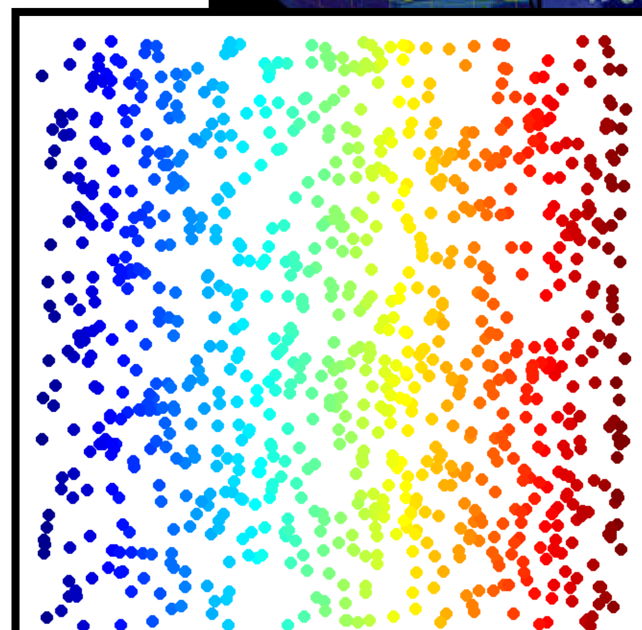
Non-linear dimension reduction.

Manifold Learning

Low dimensional shape in high dimensional space.
(ex: 2D in 3D)







Euclidean distance: ① < ②

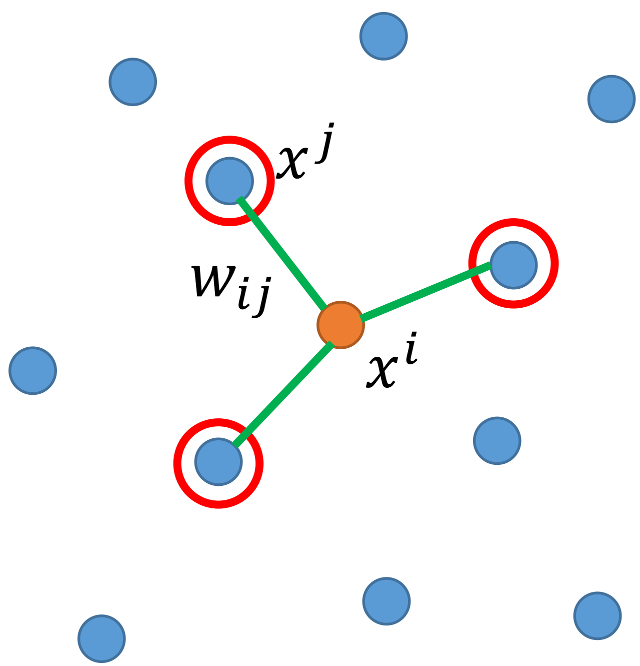
Manifold: ① > ② ← Make more sense.

Suitable for clustering or following supervised learning

Use k nearest neighbor to reduce the dimension.

Locally Linear Embedding (LLE)

Still a non-linear method.



w_{ij} represents the relation between x^i and x^j

① Find k nearest neighbor. (Find x^j)

② Find a set of w_{ij} minimizing (Find w_{ij})

Use x^j to reconstruct x^i.

$$\boxed{\sum_i \left\| x^i - \sum_j w_{ij} x^j \right\|_2}$$

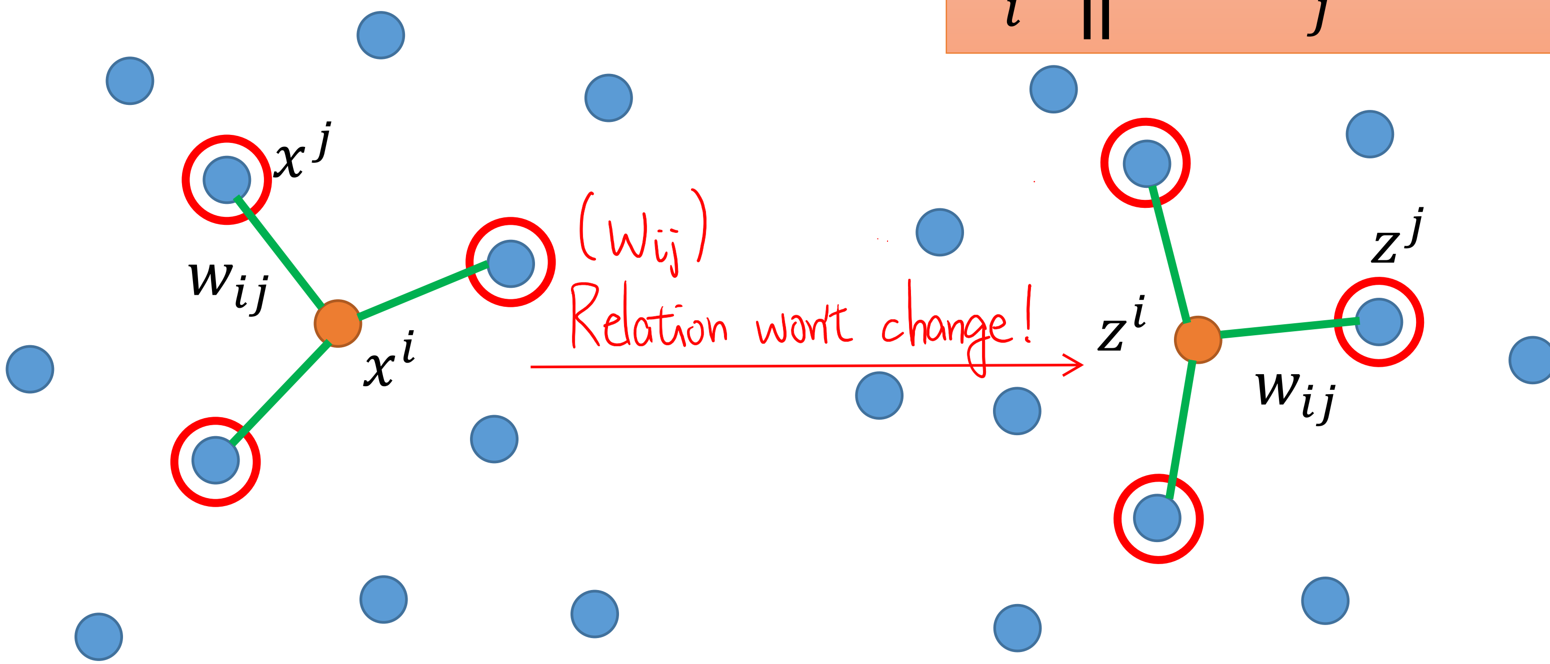
③ Then find the dimension reduction results z^i and z^j based on w_{ij} (Find z^i, z^j)

LLE

Find a set of z^i minimizing

$$\boxed{\sum_i \left\| z^i - \sum_j w_{ij} z^j \right\|_2}$$

Keep w_{ij} unchanged



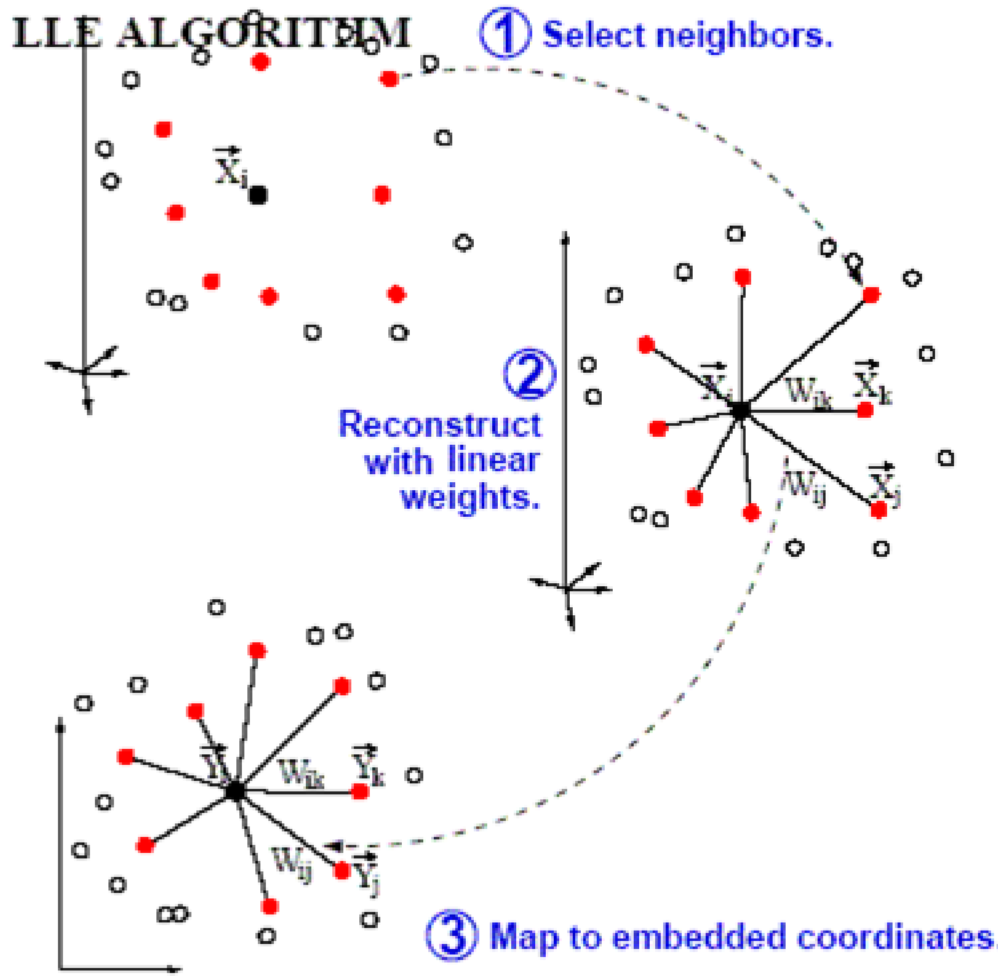
Original Space ***New (Low-dim) Space***

LLE算法认为每一个数据点都可以由其近邻点的线性加权组合构造得到。算法的主要步骤分为三步：(1)寻找每个样本点的k个近邻点；（2）由每个样本点的近邻点计算出该样本点的局部重建权值矩阵；（3）由该样本点的局部重建权值矩阵和其近邻点计算出该样本点的输出值。具体的算法流程如图2所示：

1. Compute the neighbors of each data point, $\vec{X}_i$.

2. Compute the weights W_{ij} that best reconstruct each data point $\vec{X}_i$ from its neighbors, minimizing the cost in Equation (1) by constrained linear fits.

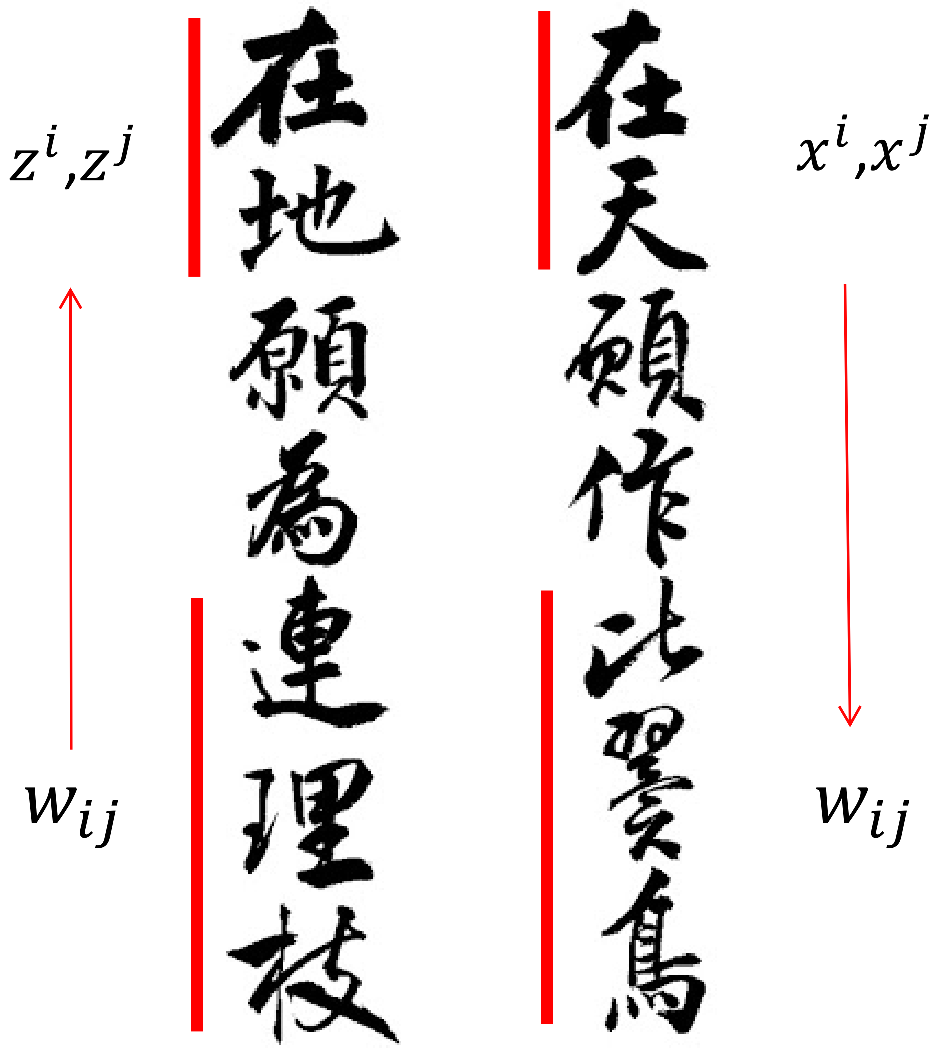
3. Compute the vectors $\vec{Y}_i$ best reconstructed by the weights W_{ij}, minimizing the quadratic form in Equation (2) by its bottom nonzero eigenvectors.

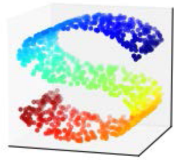


Using SVD to solve.

图 2 LLE算法步骤

LLE



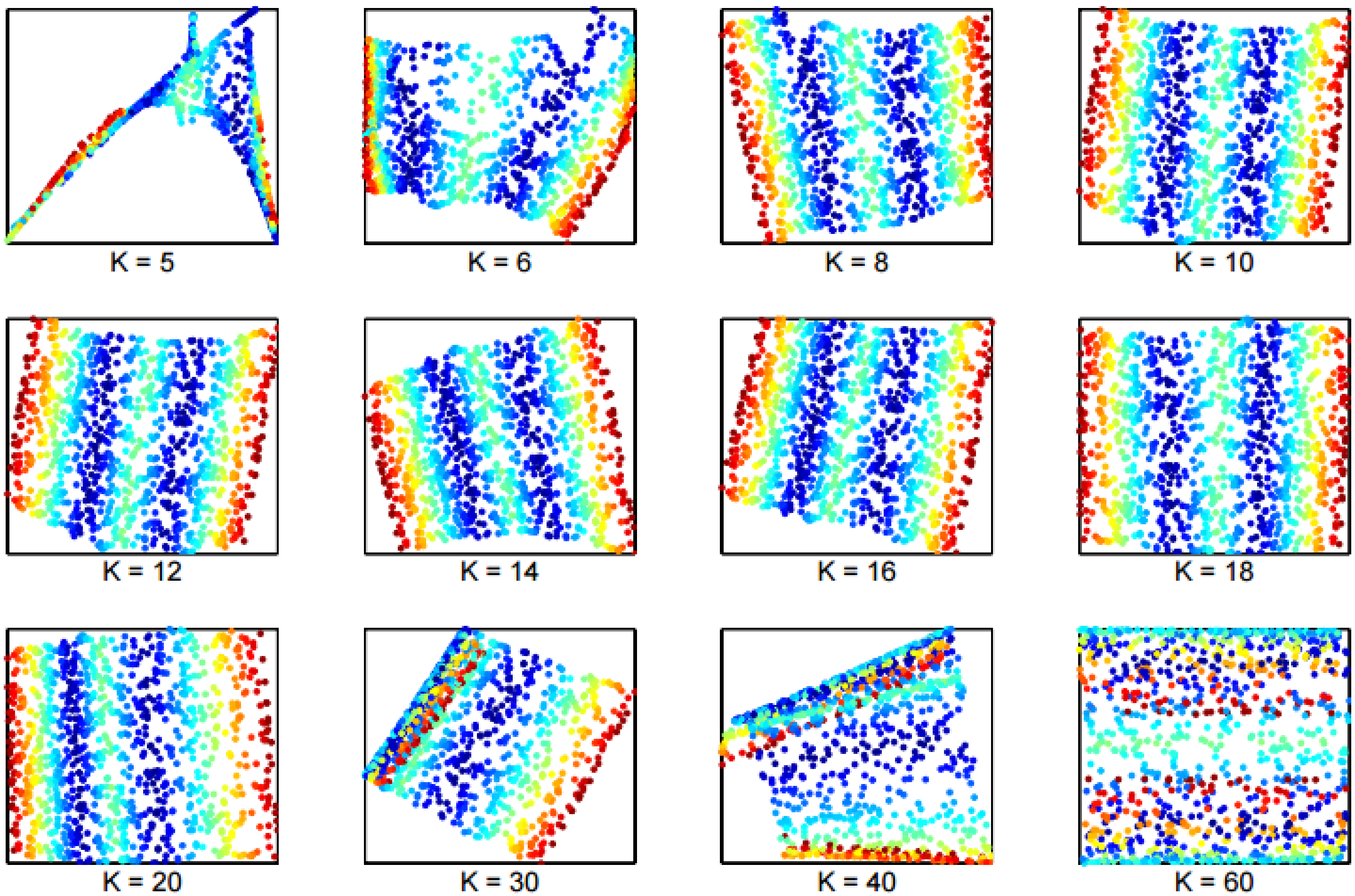
Source of image: http://feetsprint.blogspot.tw/2016/02/blog-post_29.html input: 

$x \in R^3$

$z \in R^2$

LLE

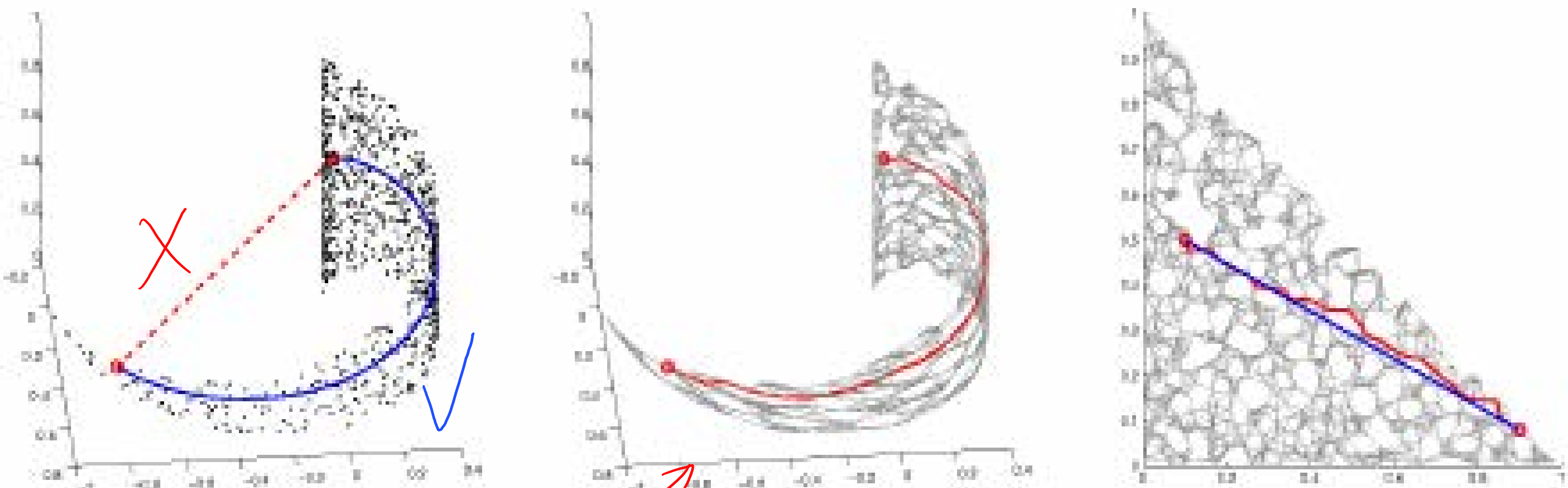
Lawrence K. Saul, Sam T. Roweis, “Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds”, JMLR, 2013



Laplacian Eigenmaps

- Graph-based approach

Distance defined by graph approximate the distance on manifold



Recall the smoothness assumption in semi-supervised learning.

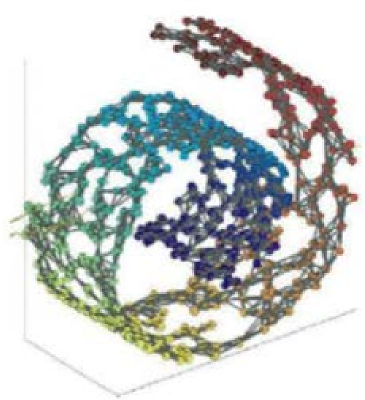
Construct the data points as a ***graph***

Laplacian Eigenmaps

The weight on the edge.

$$w_{i,j} = \begin{cases} \text{similarity} & \text{If connected} \\ 0 & \text{otherwise} \end{cases}$$

- *Review in semi-supervised learning*: If x^1 and x^2 are close in a high density region, $\hat{y}^1$ and $\hat{y}^2$ are probably the same.



$$L = \sum_{x^r} C(y^r, \hat{y}^r) \boxed{+\lambda S}$$

As a regularization term

$$S = \frac{1}{2}\sum_{i,j} w_{i,j}\left(y^i - y^j\right)^2 = \boldsymbol{y}^T L \boldsymbol{y}$$

S evaluates how smooth your label is

L: (R+U) x (R+U) matrix

Graph Laplacian

$$L = D - W$$

Laplacian Eigenmaps

- *Dimension Reduction*: If x^1 and x^2 are close in a high density region, z^1 and z^2 are close to each other.

Loss function of smoothness.

$$S = \frac{1}{2}\sum_{i,j} w_{i,j}\left(z^i - z^j\right)^2$$

Any problem? How about $z^i = z^j = \mathbf{0}$? $S=0$

Solution: Giving some constraints to z:

It should fill up the space.

If the dim of z is M, $\boxed{\text{Span}\{z^1, z^2, \dots z^N\} = R^M}$

ex: When we reduce to 2D, all points can't be on one single line or one single point.

Spectral clustering: clustering on z

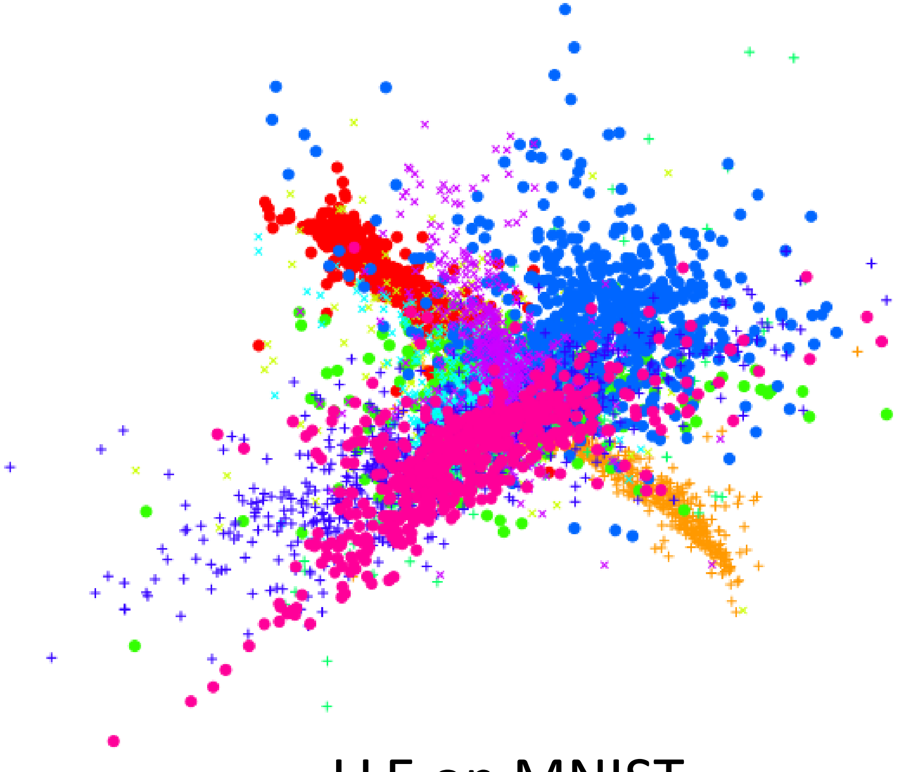
Belkin, M., Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems* . 2002

T-distributed Stochastic Neighbor Embedding (t-SNE)

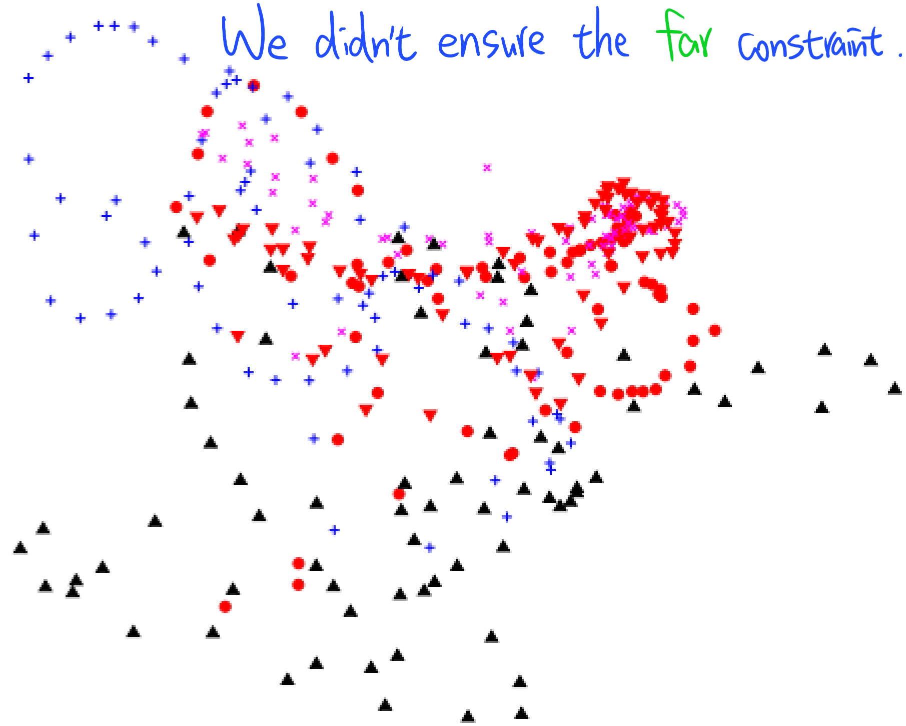
We only ensured that the points that are close to each other should be close to each other as well after the dimension reduction.

- Problem of the previous approaches
  - Similar data are close, but different data may collapse

We didn't ensure the far constraint.



LLE on MNIST



LLE on COIL-20

t-SNE

$$x \xrightarrow{\text{dimension reduction}} z$$

Before dimension reduction:

Compute similarity between all pairs of x: $S(x^i, x^j)$

After dimension reduction:

Compute similarity between all pairs of z: $S'(z^i, z^j)$

Can be different evaluation metric.

$$P(x^j|x^i) = \frac{S(x^i, x^j)}{\sum_{k \neq i} S(x^i, x^k)} \qquad Q(z^j|z^i) = \frac{S'(z^i, z^j)}{\sum_{k \neq i} S'(z^i, z^k)}$$

Normalization

(Avoid scaling between different evaluation metric.)

Find a set of z making the two distributions as close as possible

$$L = \sum_i KL\left(P(*|x^i)||Q(*|z^i)\right)$$

KL divergence

(The similarity between two distributions.)

Ensure both "close" and "far" constraints.

$$= \sum_i \sum_j P(x^j|x^i) log \frac{P(x^j|x^i)}{Q(z^j|z^i)}$$

Ignore σ for simplicity

t-SNE –Similarity Measure

Before dimension reduction:

Both SNE and t-SNE:

$$S(x^i, x^j)$$

$$= exp\left(-\left\|x^i - x^j\right\|_2\right)$$

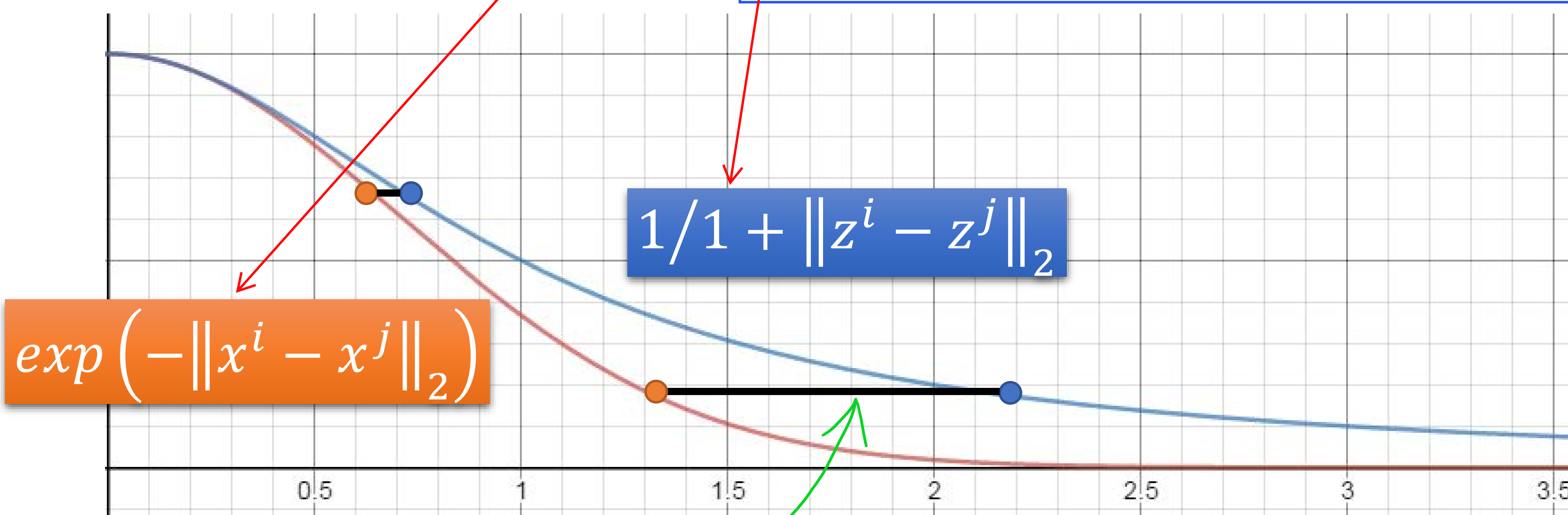
After dimension reduction:

SNE:

$$S'(z^i, z^j) = exp\left(-\left\|z^i - z^j\right\|_2\right)$$

t-SNE:

$$S'(z^i, z^j) = 1/1 + \left\|z^i - z^j\right\|_2$$

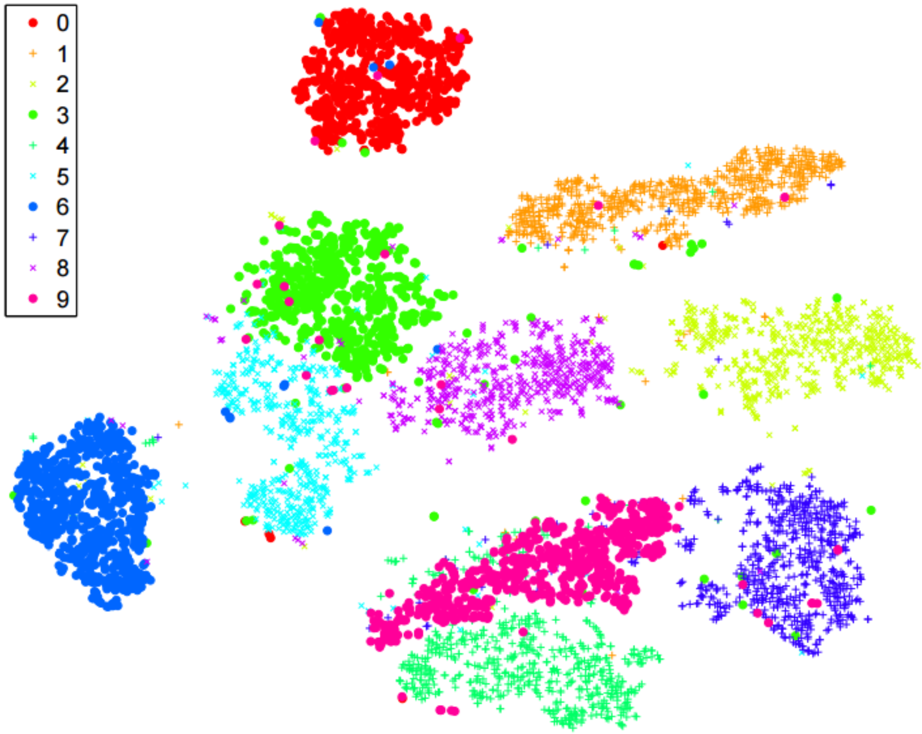


$\left\|x^i - x^j\right\|_2$, $\left\|z^i - z^j\right\|_2$

t-SNE: Exaggerate the gap.

t-SNE

- Good at visualization



t-SNE on MNIST

t-SNE on COIL-20

To learn more …

- Locally Linear Embedding (LLE): [Alpaydin, Chapter 6.11]
- Laplacian Eigenmaps: [Alpaydin, Chapter 6.12]
- t-SNE
    - Laurens van der Maaten, Geoffrey Hinton, "Visualizing Data using t-SNE", JMLR, 2008
    - Excellent tutorial: https://github.com/oreillymedia/t-SNE-tutorial