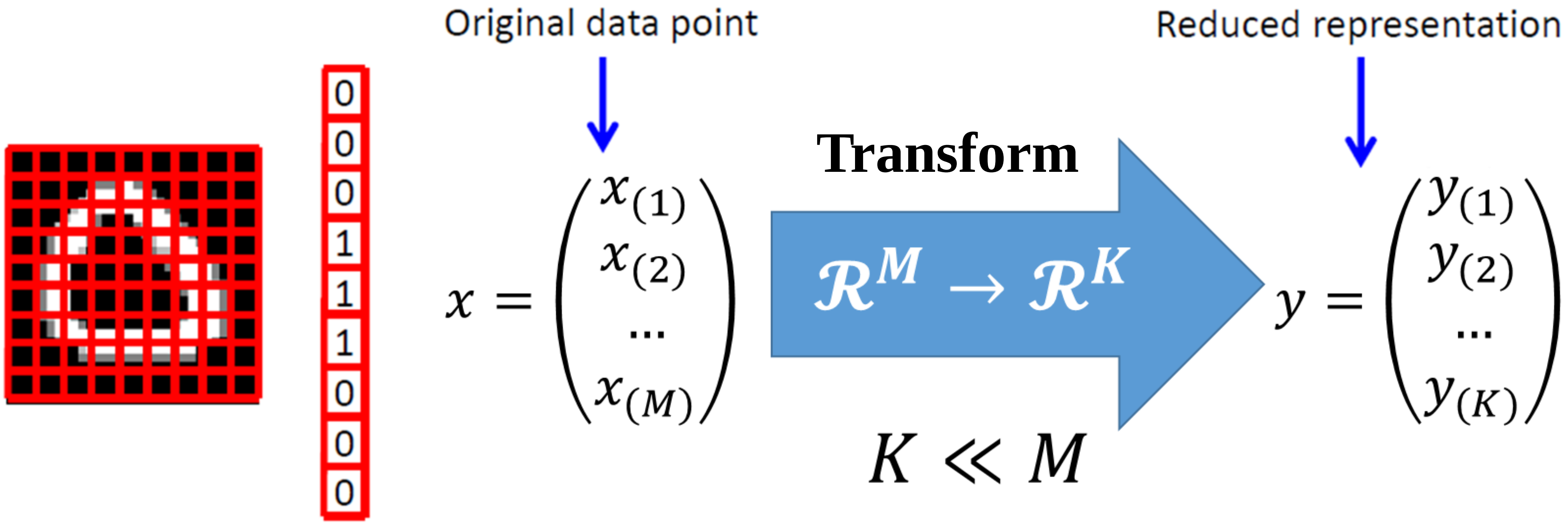
Dimensionality Reduction

Liang Liang

What is dimensionality reduction ?

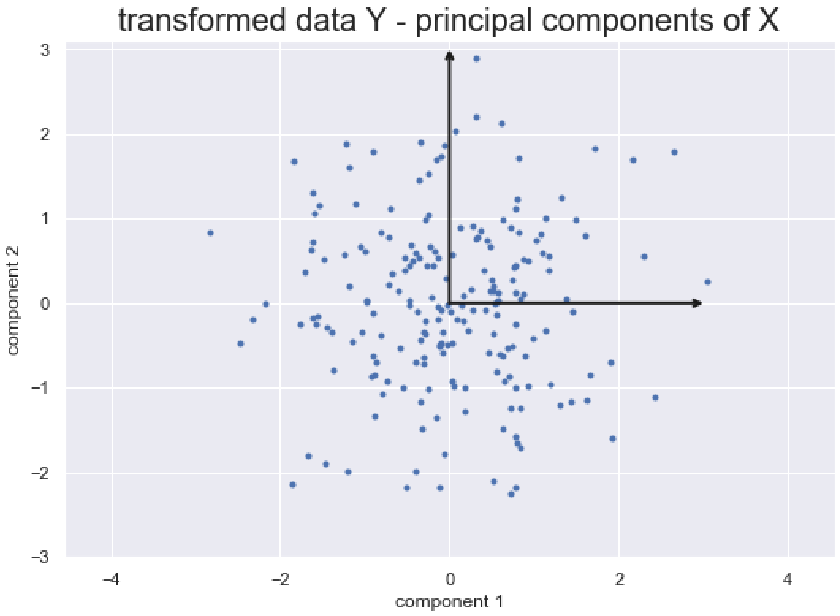


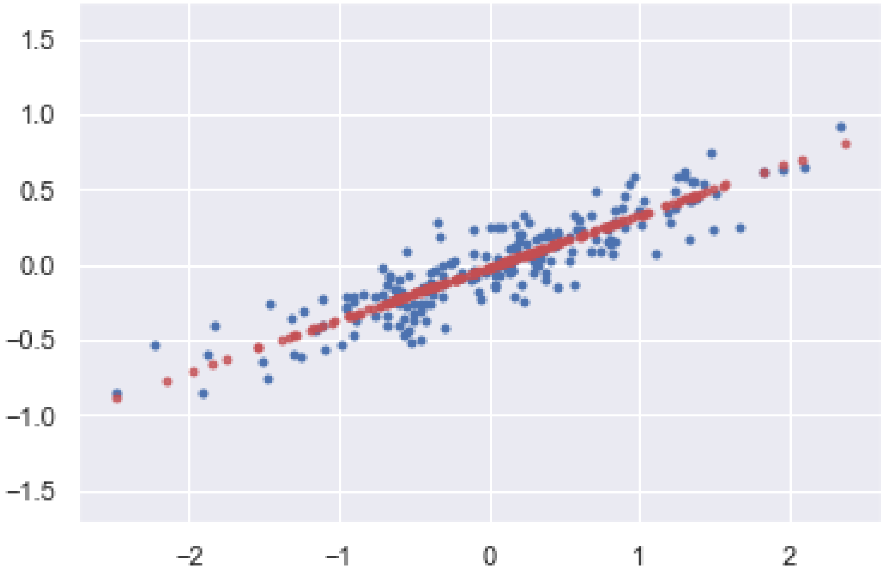
the original data point x is transformed to a new data point y in a lower dimensional space

PCA: forward transform and inverse transform

$$\boldsymbol{y_n} = A\,(\boldsymbol{x_n} - \mu)$$

forward transform (K=2)





$$\boldsymbol{\tilde{x}_n} = B\,\boldsymbol{y_n} + \mu$$

inverse transform (K=1)

$$\boldsymbol{\tilde{x}_n} \approx \boldsymbol{x_n}$$

Connect the PCA algorithm to Python Code: PCA.ipynb

Input: 200 data points $\{x_1, x_2, x_3, \dots, x_{200}\}$ and $x_n \in \mathcal{R}^2$

Python:

Each row of the 2D array X is a data point

Index starts from 0

```
print(X)
```

```
[[-6.25301618e-01 -1.70063657e-01]
 [ 9.60695033e-01  5.90900597e-01]
 [-5.98543385e-01 -4.02593393e-01]
 [-2.22805938e+00 -5.32576740e-01]
 [-4.61430060e-01 -4.98867244e-01]
 [-9.58929028e-01 -2.69331024e-01]
 [-6.73079909e-01 -3.38308547e-01]
 [ 1.30501861e+00  5.91357846e-01]
 [ 3.74545597e-01 -9.85442049e-02]
 [-1.82628627e+00 -4.06170254e-01]
 [ 6.68262284e-01  3.36877396e-01]
 [-5.82646676e-01 -1.77369217e-01]
 [-4.18128976e-01 -3.73811389e-01]
 [ 1.72209371e-01  2.64668836e-01]
 [ 3.77116687e-01  1.88442969e-01]
 [-6.79396230e-01 -1.31601978e-01]
 [ 1.03148960e+00  4.25550018e-01]
 [ 3.36041799e-01  3.90982721e-02]
 [ 7.05745985e-01  4.88730649e-01]
 [ 8.39511547e-01  1.52125872e-01]
```

Connect the PCA algorithm to Python Code

Step-1: Estimate the mean μ and covariance matrix C from the data

$$\mu = \frac{1}{200}\sum_{n=1}^{200} x_n, \qquad C = \frac{1}{200}\sum_{n=1}^{200}(x_n - \mu)(x_n - \mu)^T$$

Step-2: Compute the eigenvectors $w_1, w_2 \ldots$ of C, corresponding to the eigenvalues $\lambda_1, \lambda_2, \ldots$ and $\lambda_1 \geq \lambda_2 \ldots$

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=False)
pca.fit(X)
```

when whiten is False, output of the forward transform is β
when whiten is True, output of the forward transform is y

Multidimensional Scaling (MDS)

- **Input**: N data points $\{x_1, x_2, x_3, \ldots, x_N\}$ and $x_n \in \mathcal{R}^M$
- **Output**: N data points $\{y_1, y_2, y_3, \ldots, y_N\}$ and $y_n \in \mathcal{R}^K$, $K \leq M$
- $\boldsymbol{x_n}$ **is transformed to** $\boldsymbol{y_n}$ in a lower dimensional space, for n=1 to N

- **Objective**: find the output to minimize the so-called stress function

$$S(y_1, y_2, y_3, \ldots, y_N) = \sum_{i \neq j} \left(d(x_i, x_j) - d(y_i, y_j)\right)^2$$

$d(x_i, x_j)$ is a distance measure, e.g., Euclidean distance

which means if x_i is close to x_j, then y_i is close to y_j

x_i and y_i refer to the same object-i; x_j and y_j refer to the same object-j

textbook: elements of statistical learning

Multidimensional Scaling (MDS)

- **Objective**: find the output to minimize the so-called stress function

$$S(y_1, y_2, y_3, \ldots, y_N) = \sum_{i \neq j} \left(d(x_i, x_j) - d(y_i, y_j)\right)^2$$

- $d(x_i, x_j)$ is a distance measure, e.g., Euclidean distance
- **Input:** pairwise distance matrix

$$\begin{bmatrix} 0 & d(x_1, x_2) & \ldots & d(x_1, x_N) \\ d(x_2, x_1) & 0 & \ldots & d(x_2, x_N) \\ \vdots & \vdots & \vdots & \vdots \\ d(x_N, x_1) & d(x_N, x_2) & \ldots & 0 \end{bmatrix}$$

the matrix can be visualized as an image

MDS only needs this matrix as input

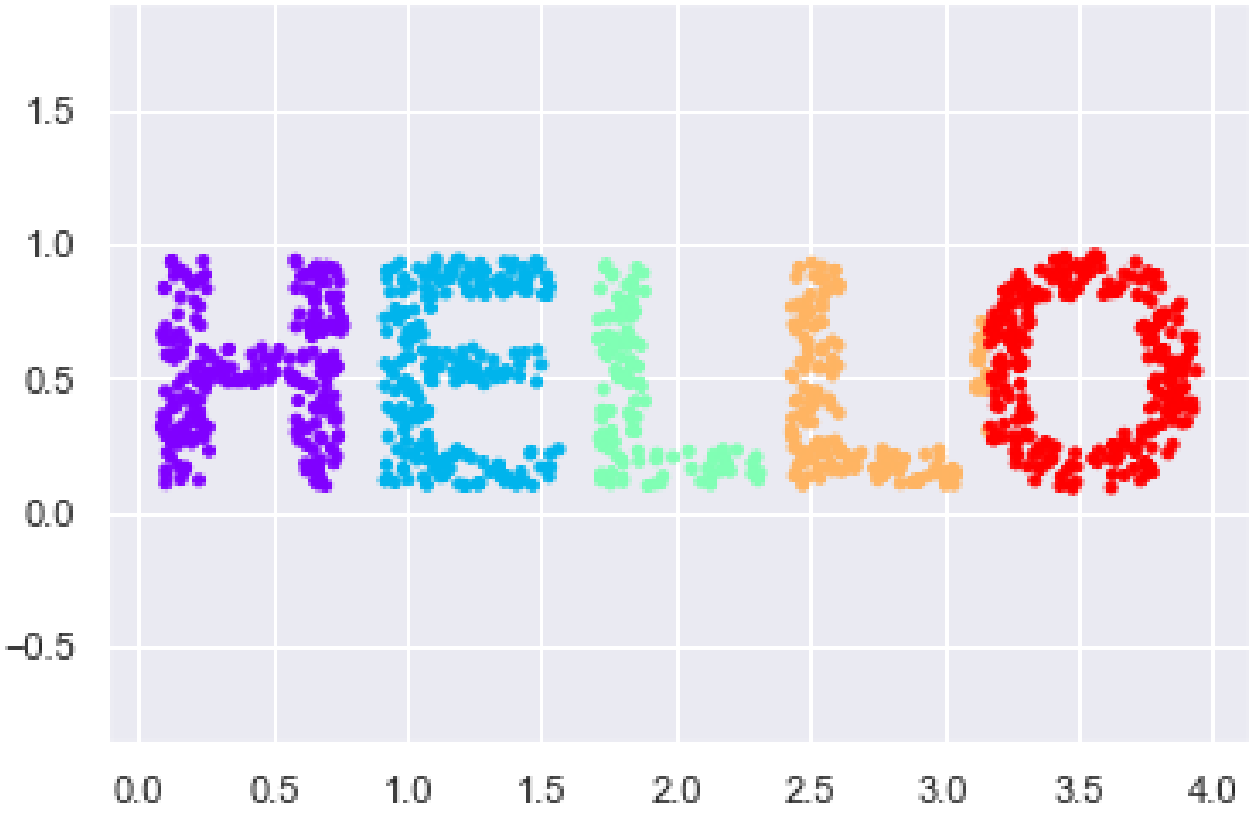
textbook: elements of statistical learning

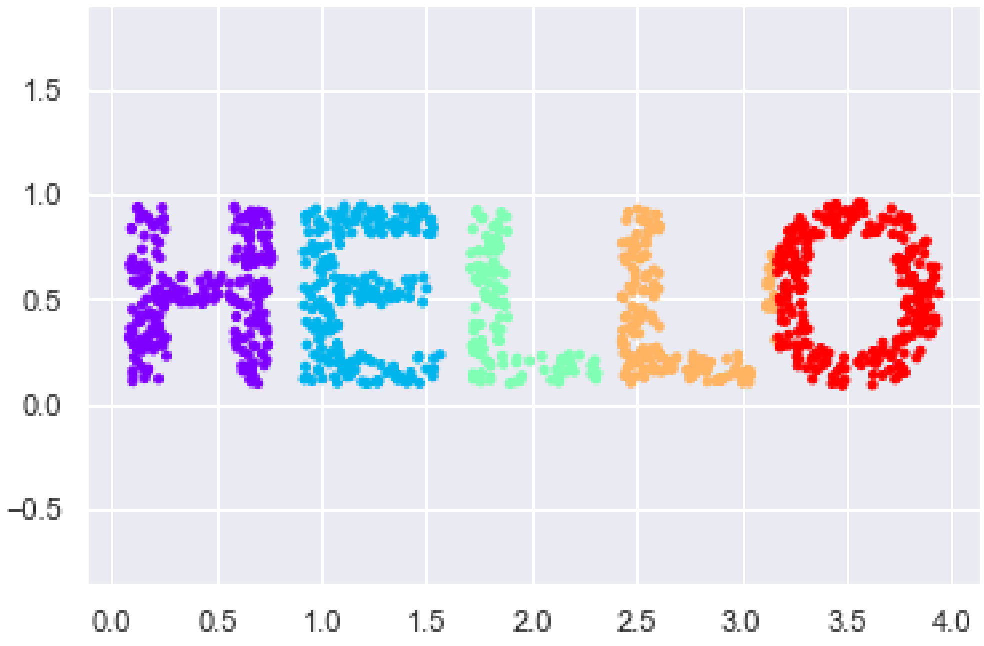
The test example "HELLO"

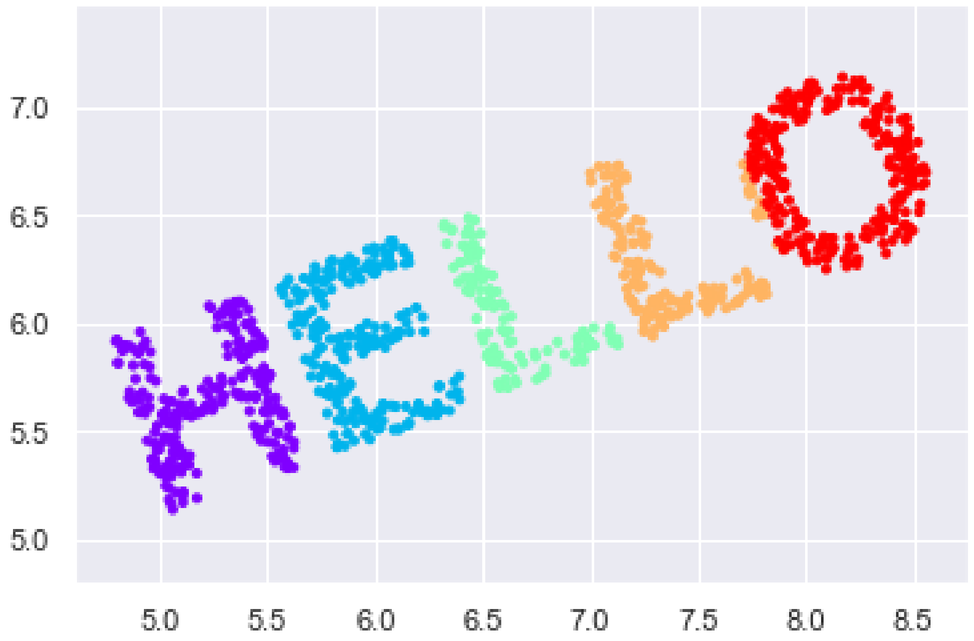
```python
def make_hello(N=1000, rseed=42):
    # Make a plot with "HELLO" text; save as PNG
```

```python
X = make_hello(1000)
colorize = dict(c=X[:, 0], cm
plt.scatter(X[:, 0], X[:, 1],
```

Each row of the 2D array **X** is a data point.

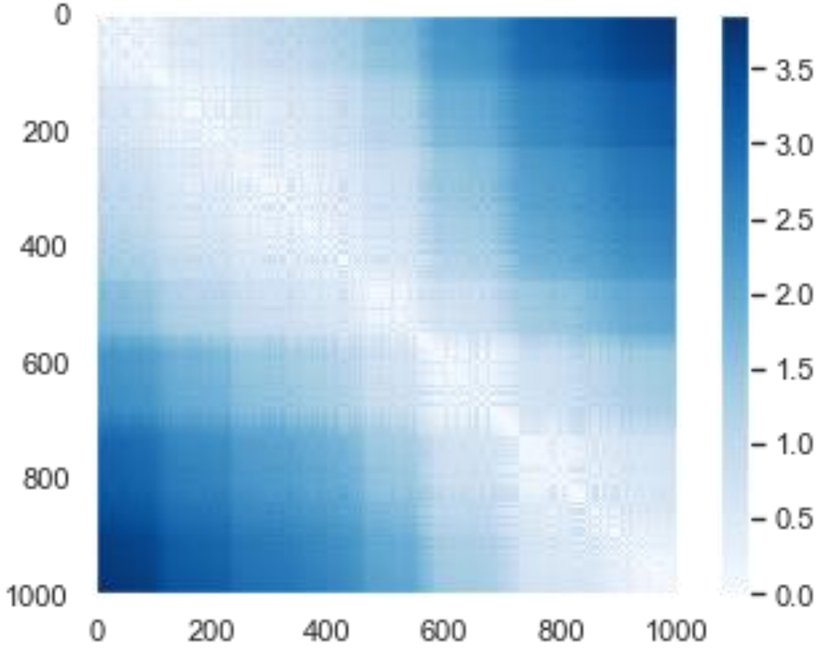




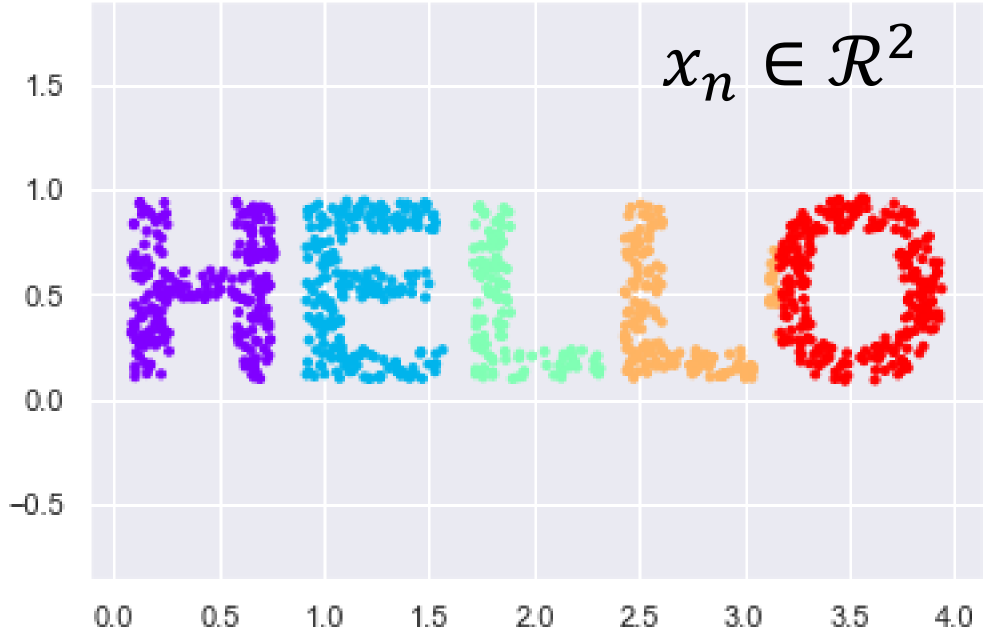


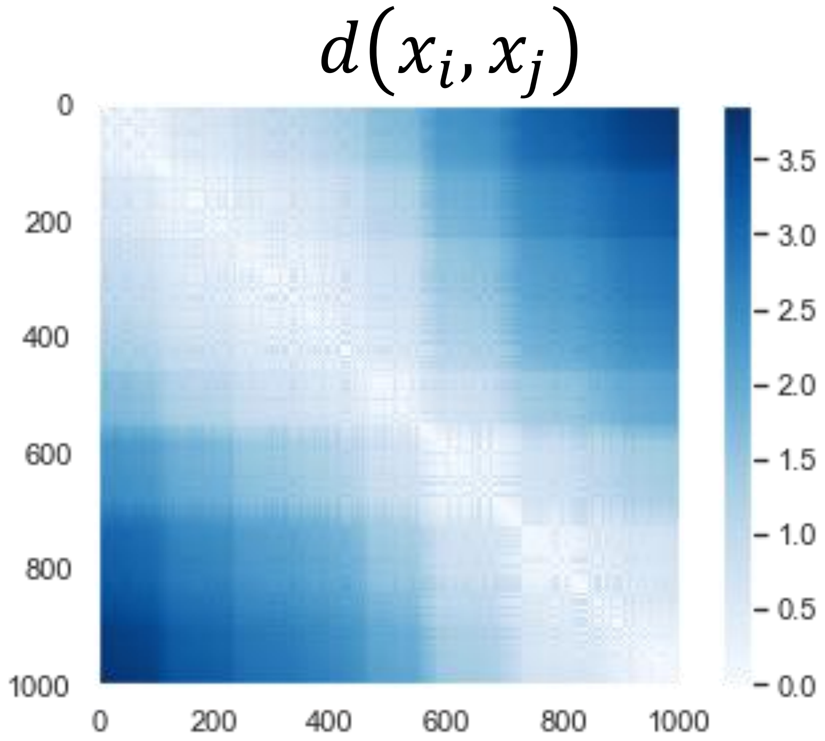
pairwise distance matrix

$$\begin{bmatrix} 0 & d(x_1, x_2) & \dots & d(x_1, x_N) \\ d(x_2, x_1) & 0 & \dots & d(x_2, x_N) \\ \vdots & \vdots & \vdots & \vdots \\ d(x_N, x_1) & d(x_N, x_2) & \dots & 0 \end{bmatrix}$$

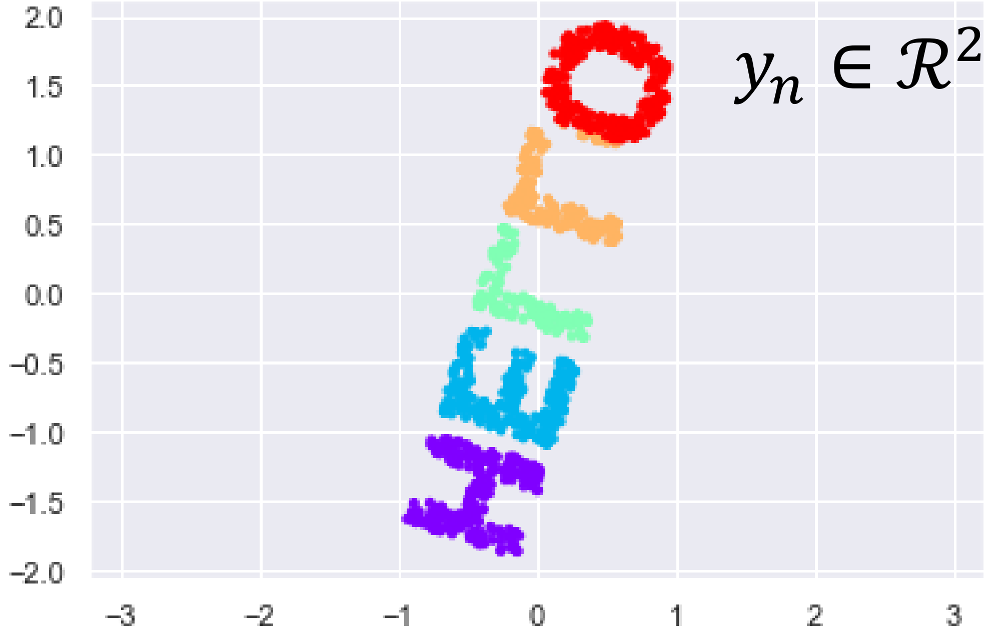


The pairwise distance matrix does not change after rotation / translation



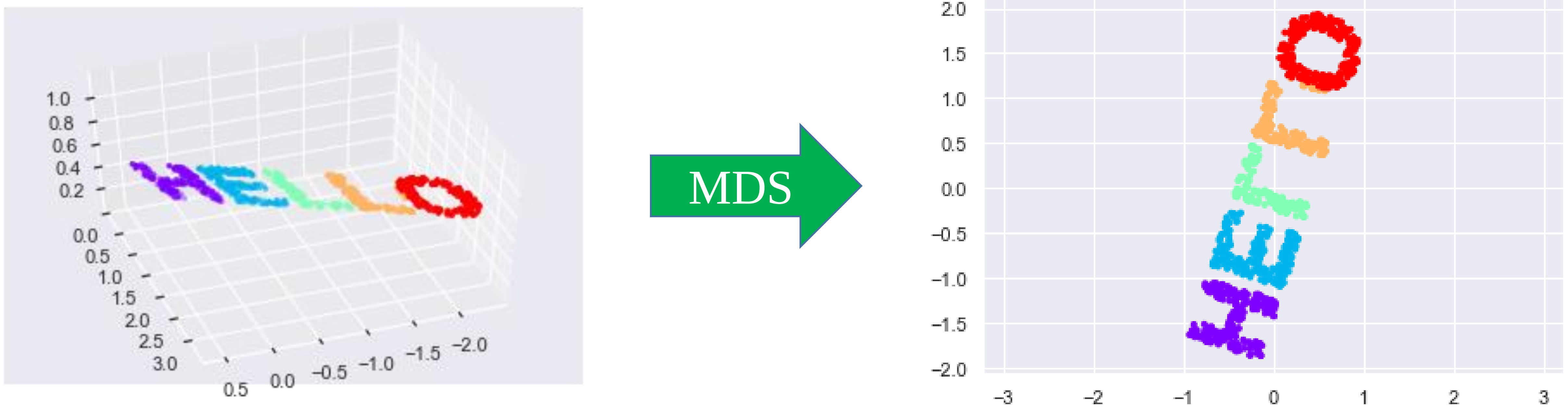


MDS



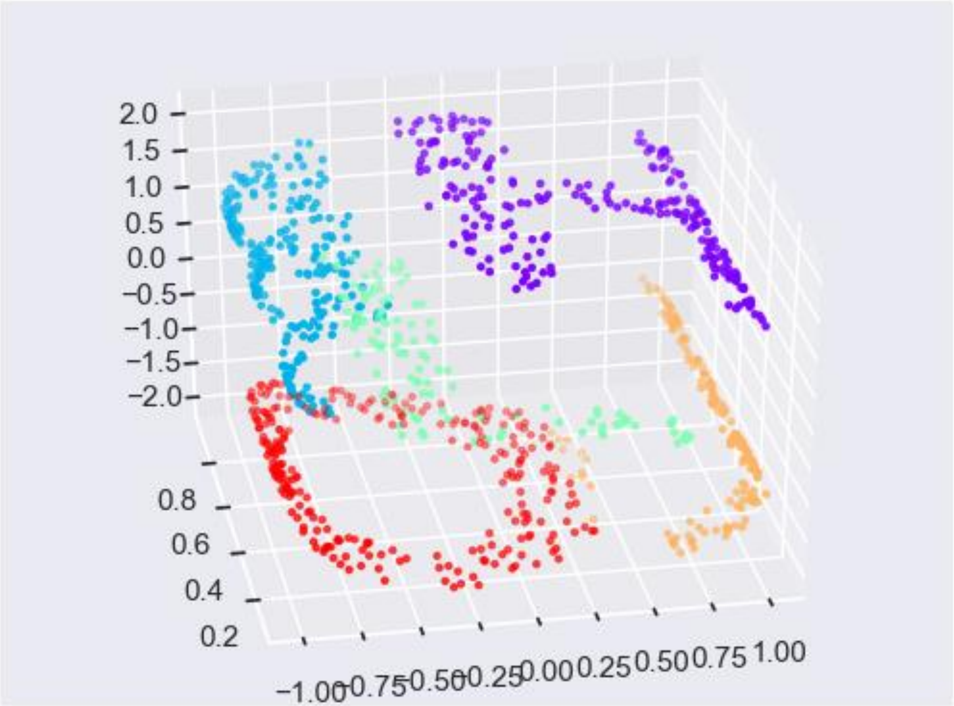
$$S(y_1, \ldots y_N) = \sum_{i \neq j} \left(d(x_i, x_j) - d(y_i, y_j) \right)^2$$

$x_n \in \mathcal{R}^3$ $\qquad$ $y_n \in \mathcal{R}^2$



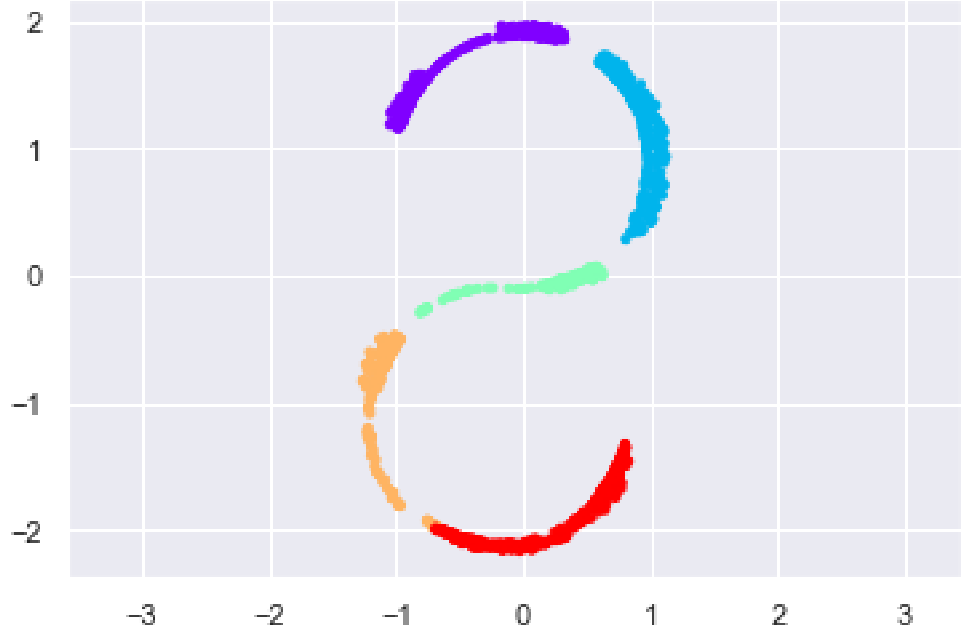
Although the data points are in 3D space, they actually live in a 2D plane.

$x_n \in \mathcal{R}^3$



"Hello" in a S-surface in 3D

$y_n \in \mathcal{R}^2$



"Hello" becomes "S" in 2D

MDS can not handle nonlinear spatial distribution

A good algorithm should be able to unwrap the S-surface

Locally linear embedding (LLE)

- For each input data point x_i, we find its K *nearest neighbors*, $\mathcal{N}(i)$, using Euclidean distance.
- Approximate each point x_i by a linear combination of its neighbors

$$\min_{w_{i,j}} \left\| x_i - \sum_{j \in \mathcal{N}(i)} w_{i,j} x_j \right\| \text{ where } \sum_{j \in \mathcal{N}(i)} w_{i,j} = 1$$

- Find output data point y_i such that it can also be approximated by its neighbors using the same set of weights $\{w_{i,j}\}$

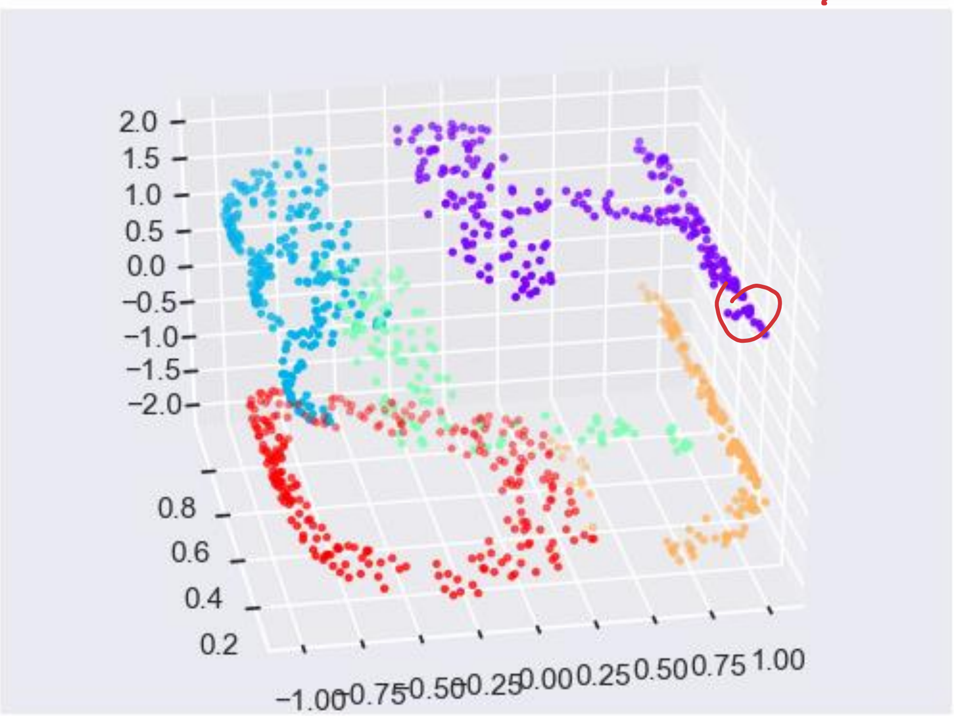
$$\min_{y} \left\| y_i - \sum_{j \in \mathcal{N}(i)} w_{i,j} y_j \right\|$$

- LLE: local neighborhood structure is preserved after the transform $\mathcal{R}^M \to \mathcal{R}^K$

textbook: elements of statistical learning

$x_n \in \mathcal{R}^3$

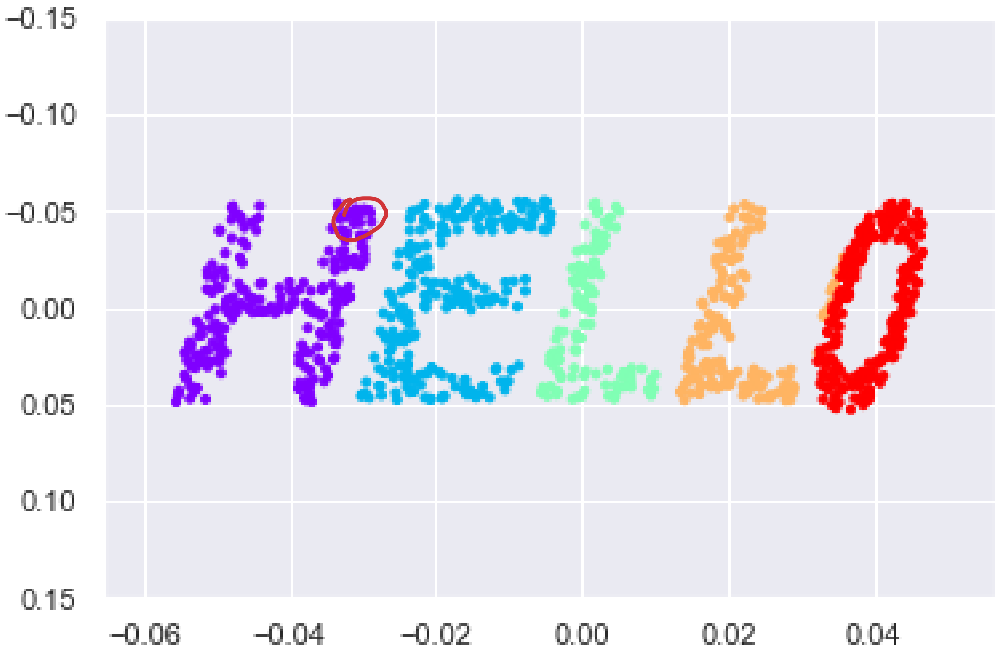
3D



LLE

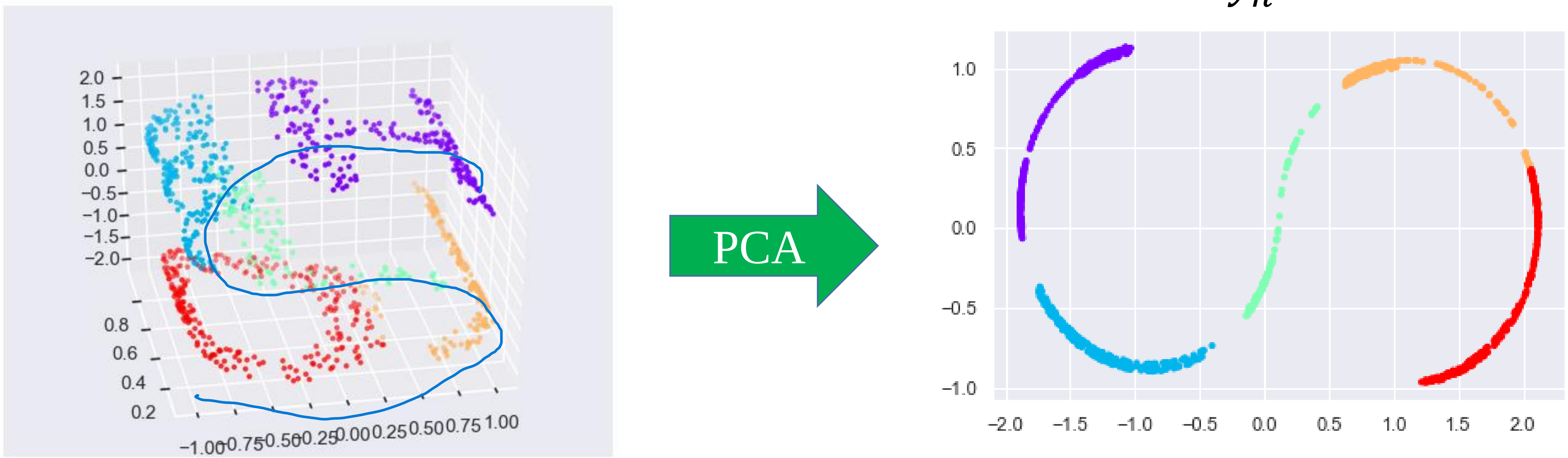
2D

$y_n \in \mathcal{R}^2$



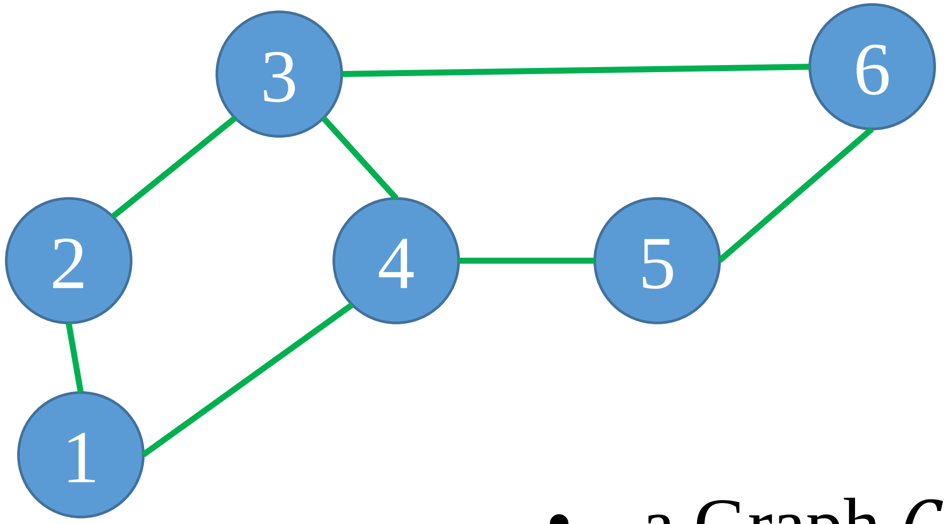
LLE is able to unwrap the S-surface

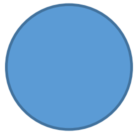
$x_n \in \mathcal{R}^3$ $\qquad$ $y_n \in \mathcal{R}^2$



PCA can not unwrap the S-surface because it is a linear transform
But, it can show the major variations of the data in 2D

Graph





a node

a link/edge between two nodes

- a Graph $G = \{V, E\}$
- V denotes the set of nodes
- E denotes the set of links/edges

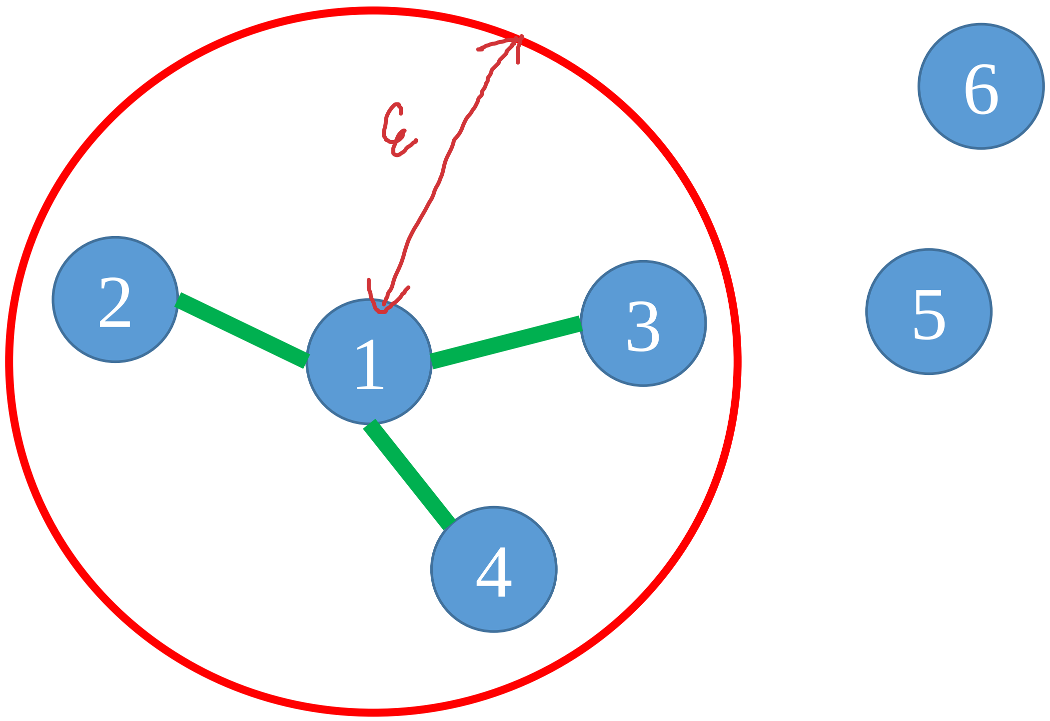
Construct an ϵ-**Neighbor Graph** from a set of data points

epsilon

Add a link between two data points/nodes
if the distance $d(x_i, x_j) \leq \epsilon$ (define the circle)

a node represents a data point

a link/edge between two nodes

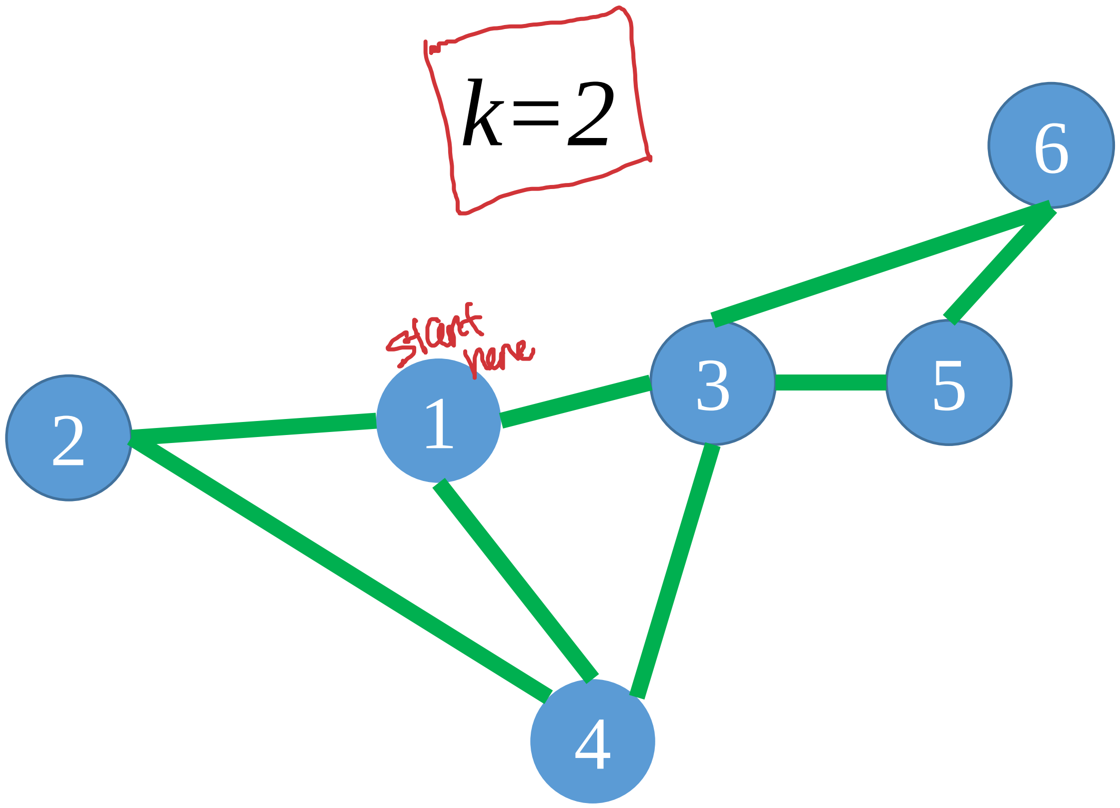


Perform the linking operation for every data point that is the center of a circle

*3 neighbors

Construct a ***k*-Nearest Neighbor Graph** from a set of data points

- Add a link between x_i and x_j if x_i is one of the k-nearest neighbors of x_j
- Perform the linking operation until there are no more links to be added

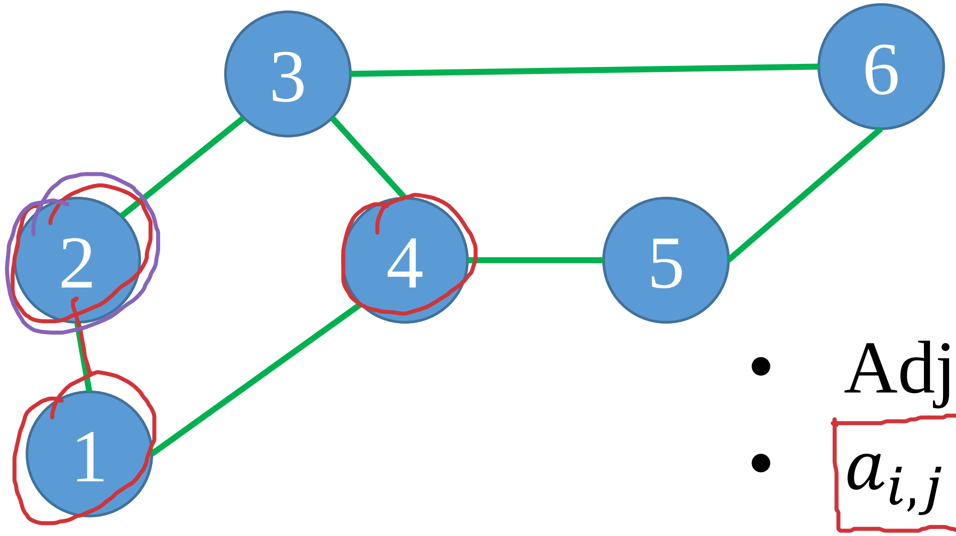


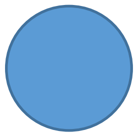
a node represents a data point

a link/edge between two nodes

We need a distance measure to define what is "nearest"

Graph - adjacency matrix



 a node represents a data point

a link/edge between two nodes

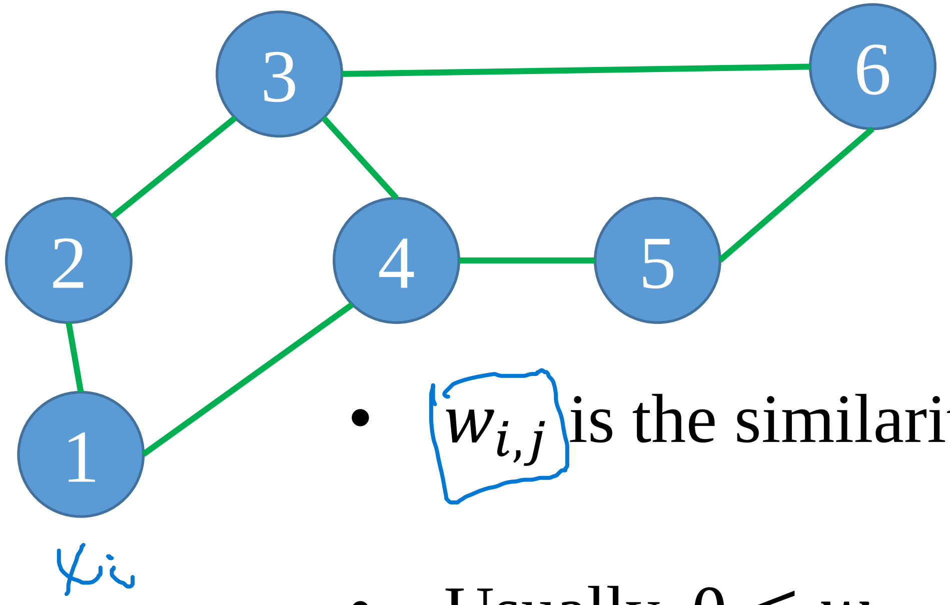
- Adjacency Matrix $A = [a_{i,j}]$
- $a_{i,j} = 1$ if node-i and node-j are connected by an edge
- $a_{i,j} = 0$ if there is no edge between node-i and node-j

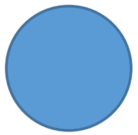
as in CSC317

$$
\begin{array}{ll}
\text{node} & \begin{array}{cccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{array} \\
A = & \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{array}
\end{array}
$$

Sometimes, Adjacency Matrix is also called Affinity Matrix

Graph - similarity matrix





a node represents a data point

a link/edge between two nodes

- $\boxed{w_{i,j}}$ is the similarity between node-i and node-j, $w_{i,j} = w_{j,i}$
- Usually, $0 \le w_{i,j} \le 1$, set $w_{i,i} = 0$. $\boxed{w_{i,j} = e^{-\gamma\left(d(x_i, x_j)\right)^2}}$

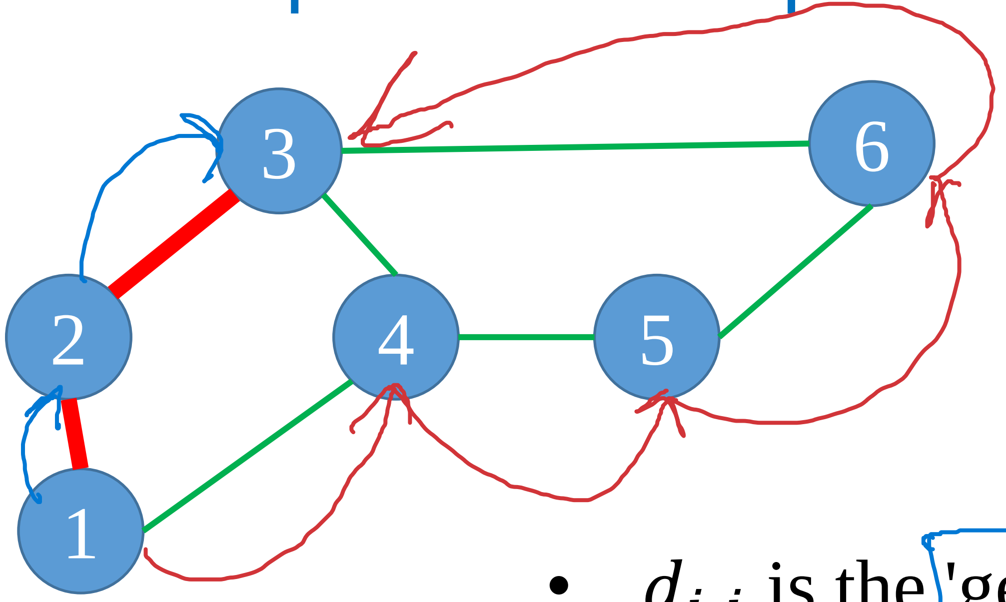
  some number!

  $d(x_i,\ x_j)$ is the distance between node-i and node-j, scalar $\gamma > 0$
- $w_{i,j} = 0$ if the edge(i, j) does not exist

  e.g. there is not link between node-6 and node-4

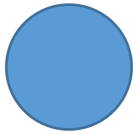
  → sklearn
- The matrix $W = [w_{i,j}]$ is called similarity/$\boxed{\text{affinity}}$ matrix

Graph - shortest path



too far

good!



a node represents a data point

a link/edge between two nodes

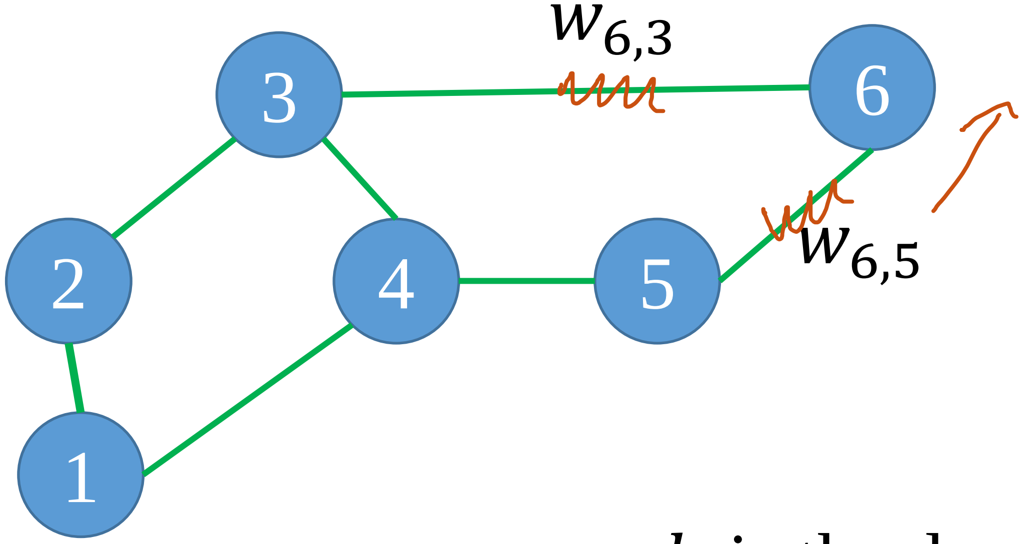
- $d_{i,j}$ is the 'geodesic distance' between node-i and node-j, which is the length of the shortest path

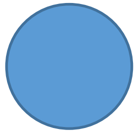
the red line shows the shortest path from node-1 to node-3

$$\boxed{d_{1,3} = d_{1,2} + d_{2,3}}$$

Graph - degree matrix

$d_6 = w_{6,3} + w_{6,5}$ (geodesic distance)



 a node represents a data point

 a link/edge between two nodes

- d_i is the degree of node-i

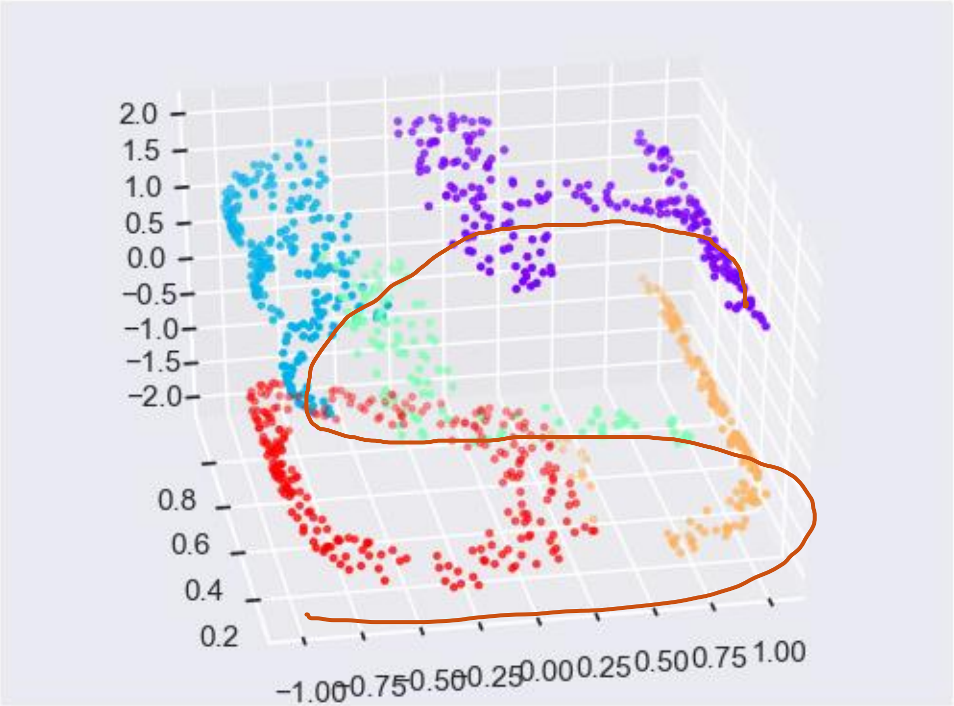
$$d_i = \sum_j w_{i,j}$$

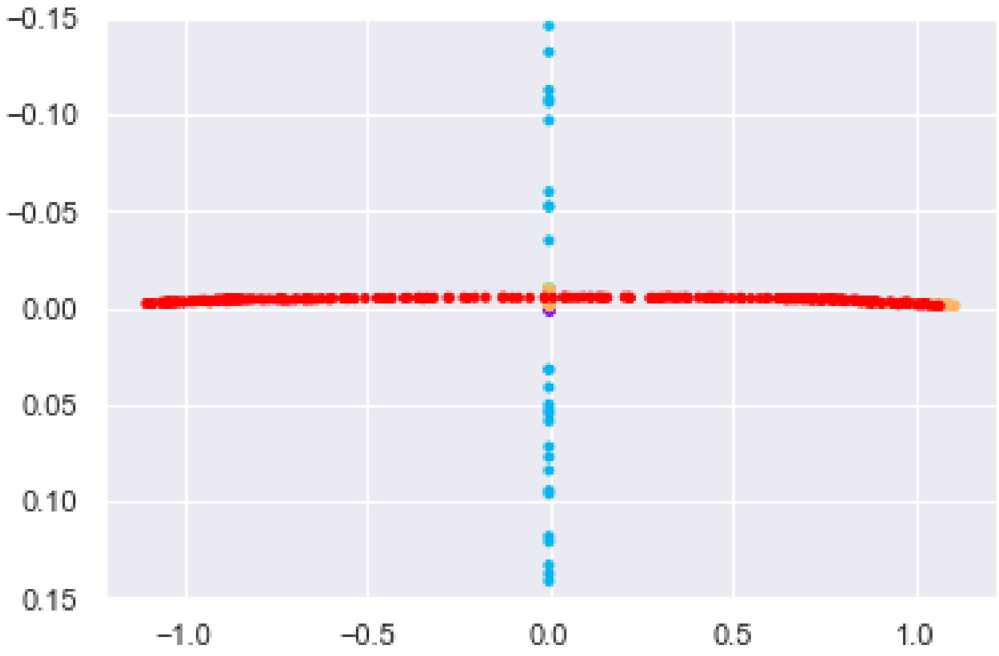
The degree matrix $D = diag(d_1, d_2, \dots, d_N)$ = $\begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_m \end{bmatrix}$

Isomap for dimensionality reduction

- Input: N data points $\{x_1, x_2, x_3, \ldots, x_N\}$ and $x_n \in \mathcal{R}^M$
- **Step-1**: build a neighbor graph of the data points
- **Step-2**: for each pair of nodes, compute $d_{i,j}$, length of the shortest path

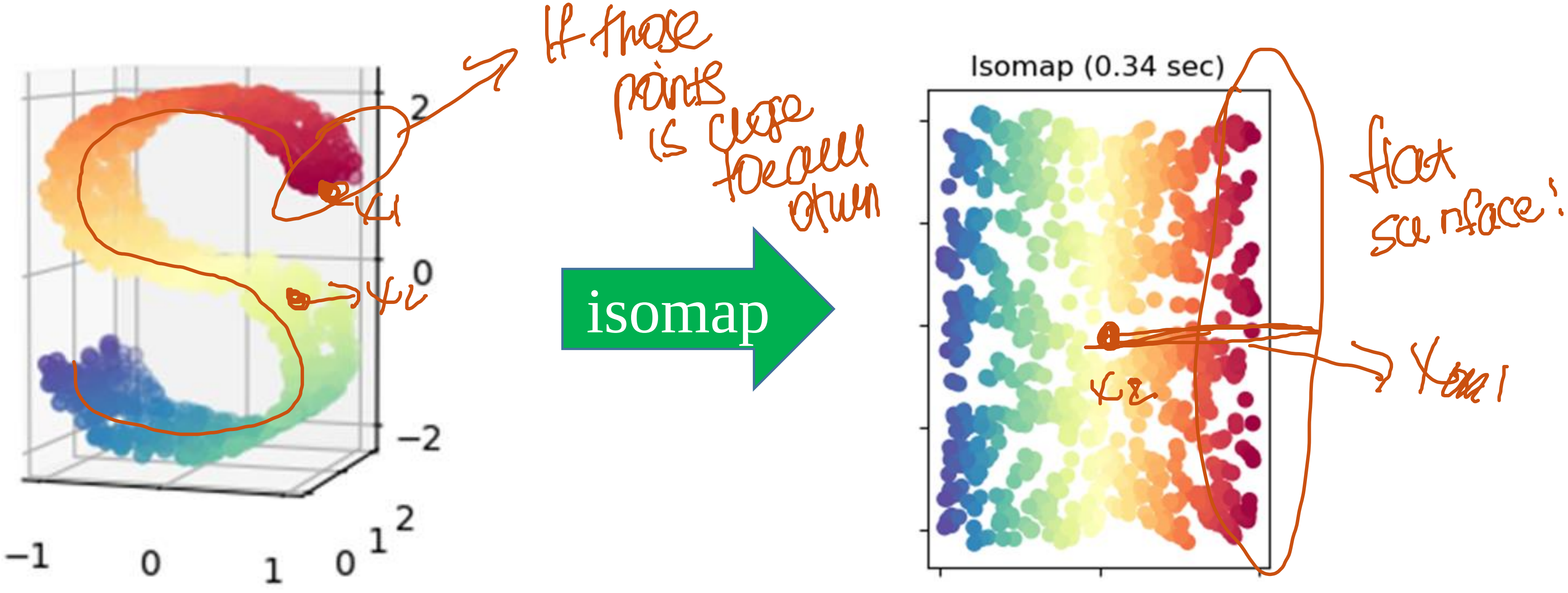
  we now have the new **distance measure** between data points
- **Step-3**: run the Multidimensional Scaling (MDS) algorithm to get the output data points points $\{y_1, y_2, y_3, \ldots, y_N\}$ and $y_n \in \mathcal{R}^K$, $K \leq M$





not a great result!

isomap does not work for those 3D points (sparse) …

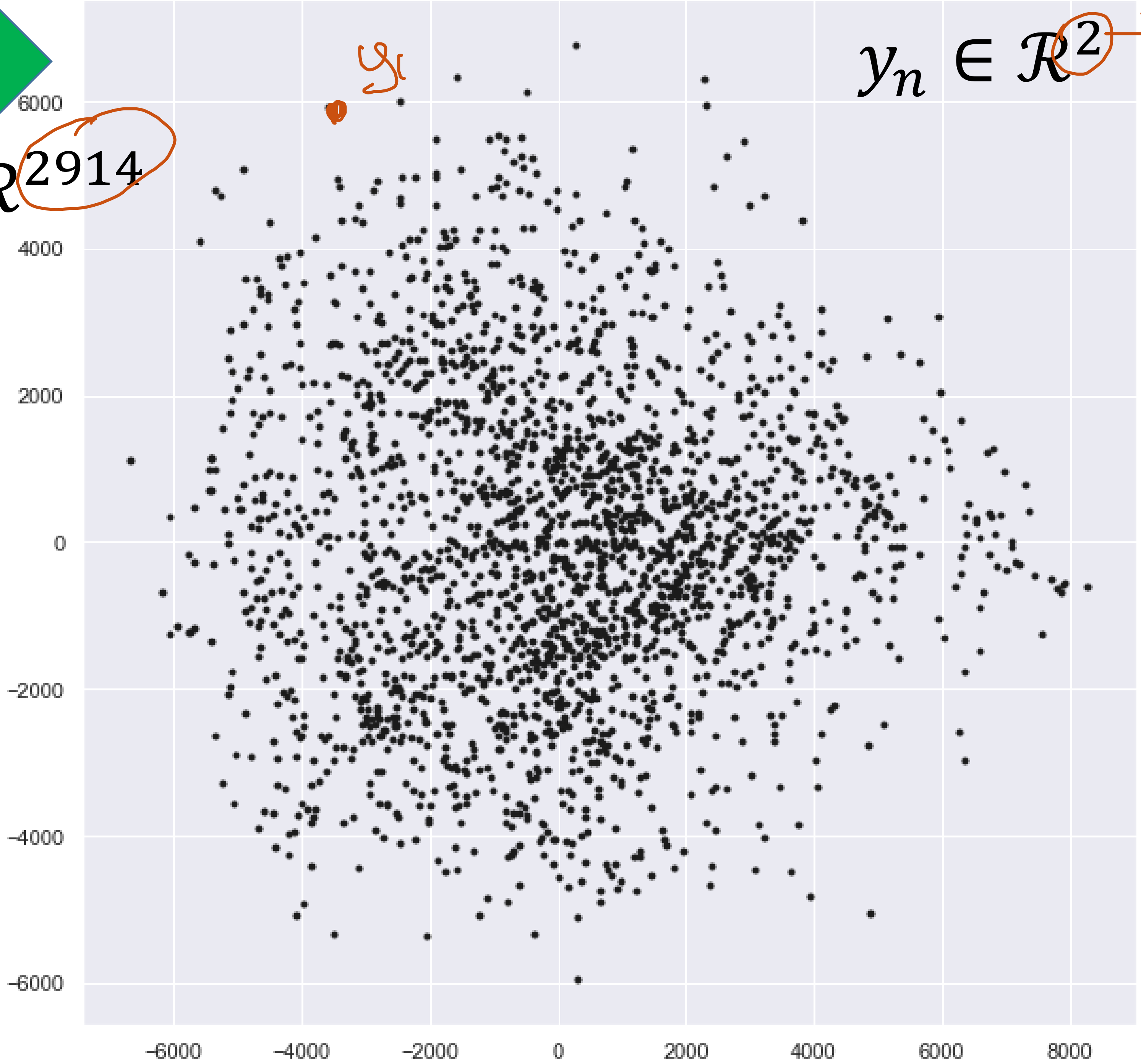


isomap works for those 3D points (dense) …

https://scikit-learn.org/stable/modules/manifold.html data points/images $\{x_1, x_2, x_3, \dots, x_N\}$ and $x_n \in \mathcal{R}^{2914}$ → Important dimension!

isomap

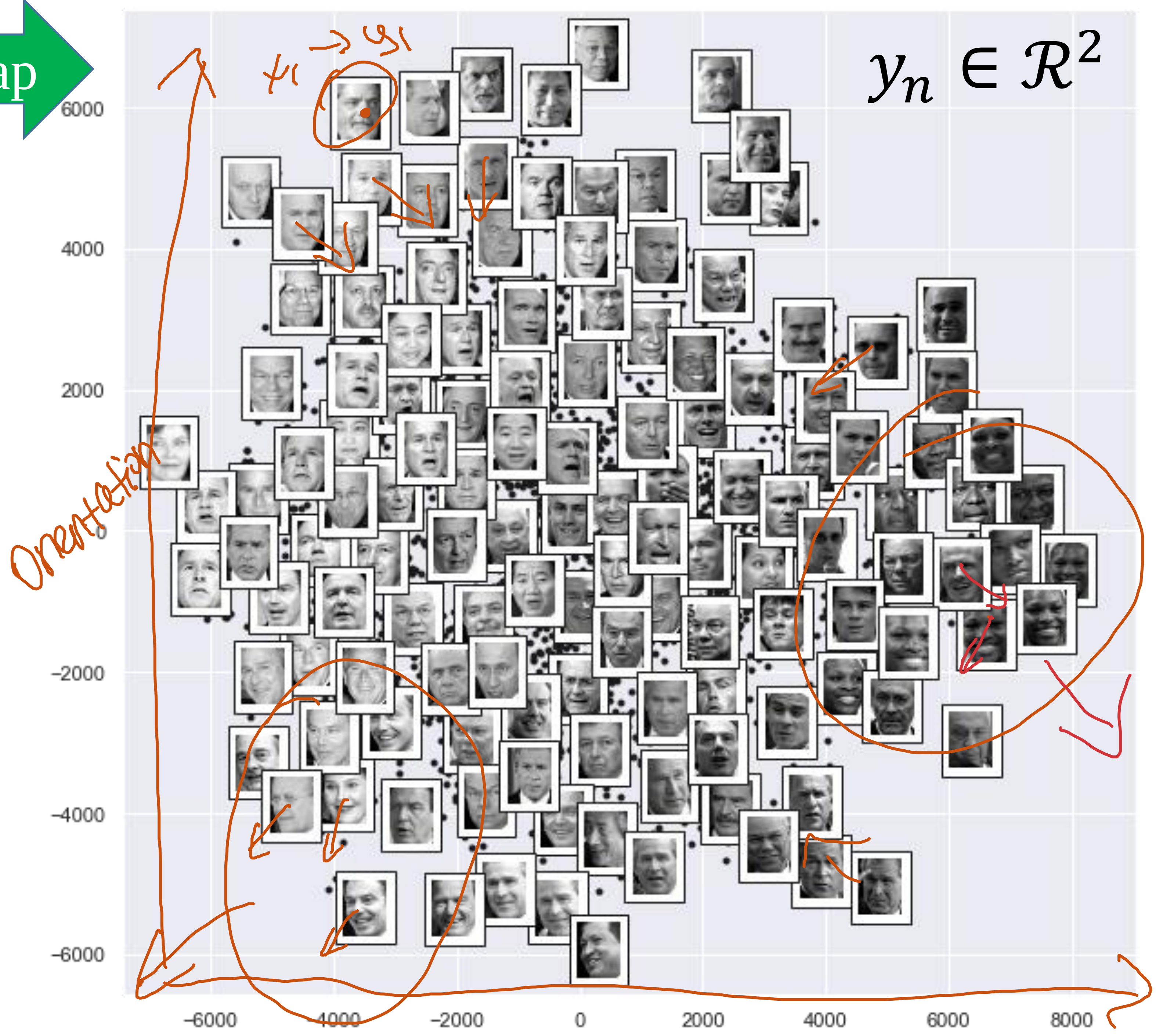
$x_n \in \mathcal{R}^{2914}$



$y_n \in \mathcal{R}^{2}$ → after transform

Next, we will show images along with the dots in 2D

isomap



$y_n \in \mathcal{R}^2$

at the centre we will plot the Image

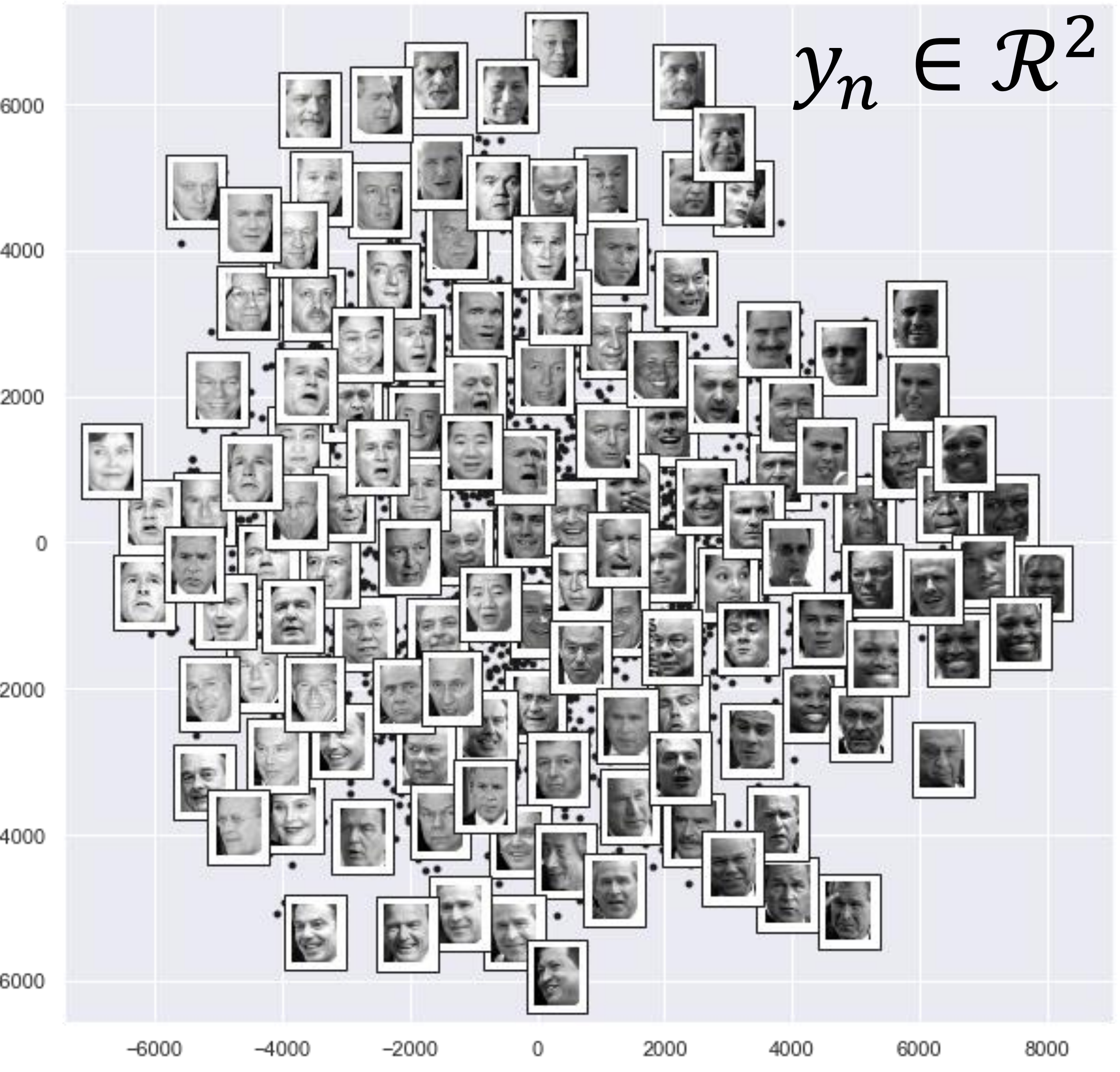
Each image is represented by two numbers/features: the overall darkness or lightness of the image from left to right, and the general orientation of the face from bottom to top.

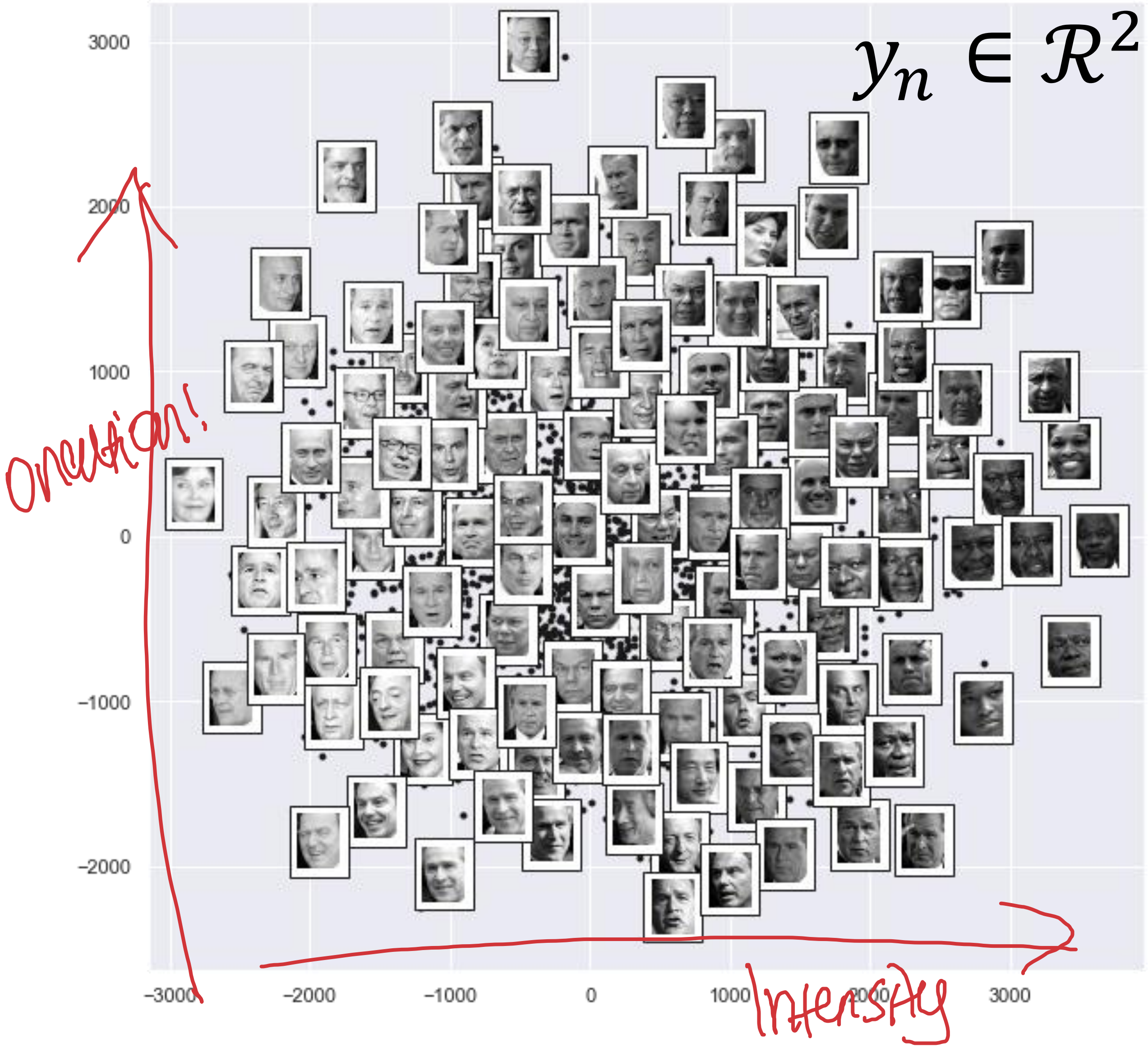
Isomap can group the data!

isomap

$$x_n \in \mathcal{R}^{2914}$$

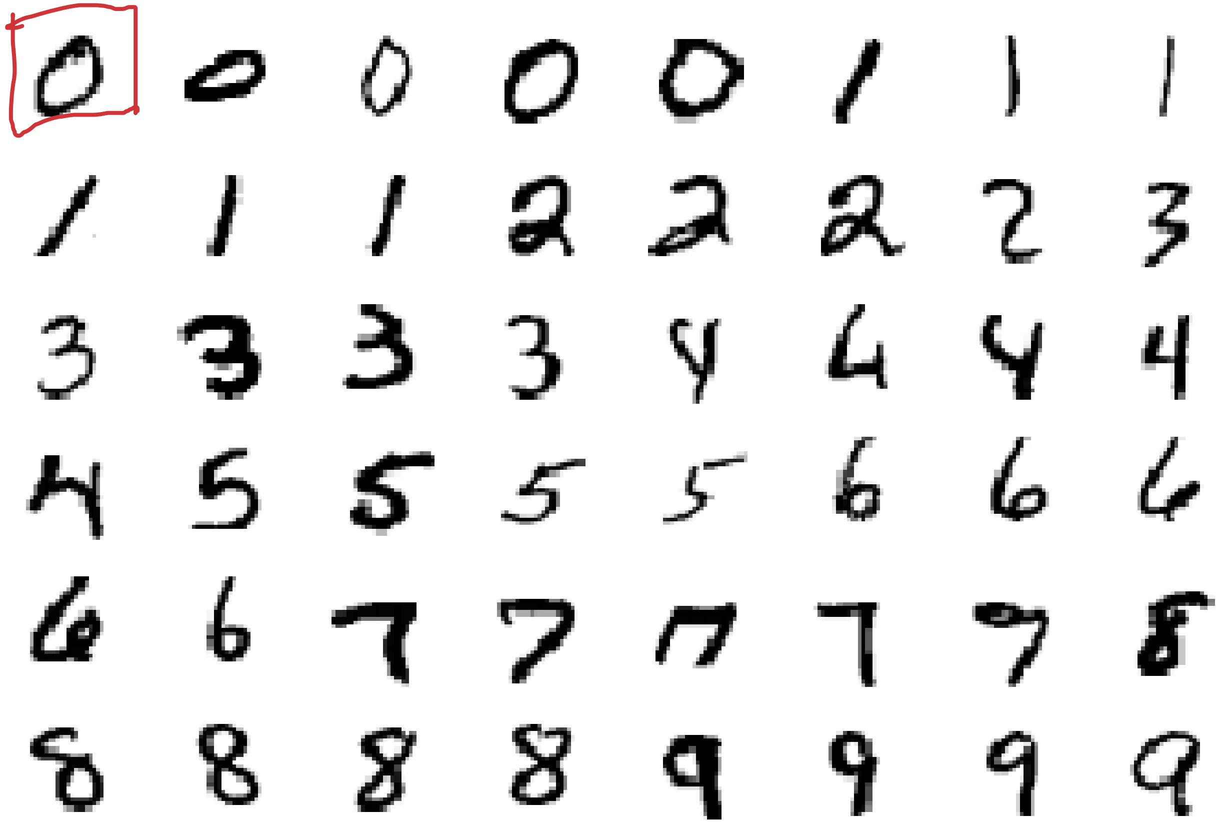
PCA





data points/images $\{x_1, x_2, x_3, \dots, x_N\}$ and $x_n \in \mathcal{R}^{\boxed{784}}$

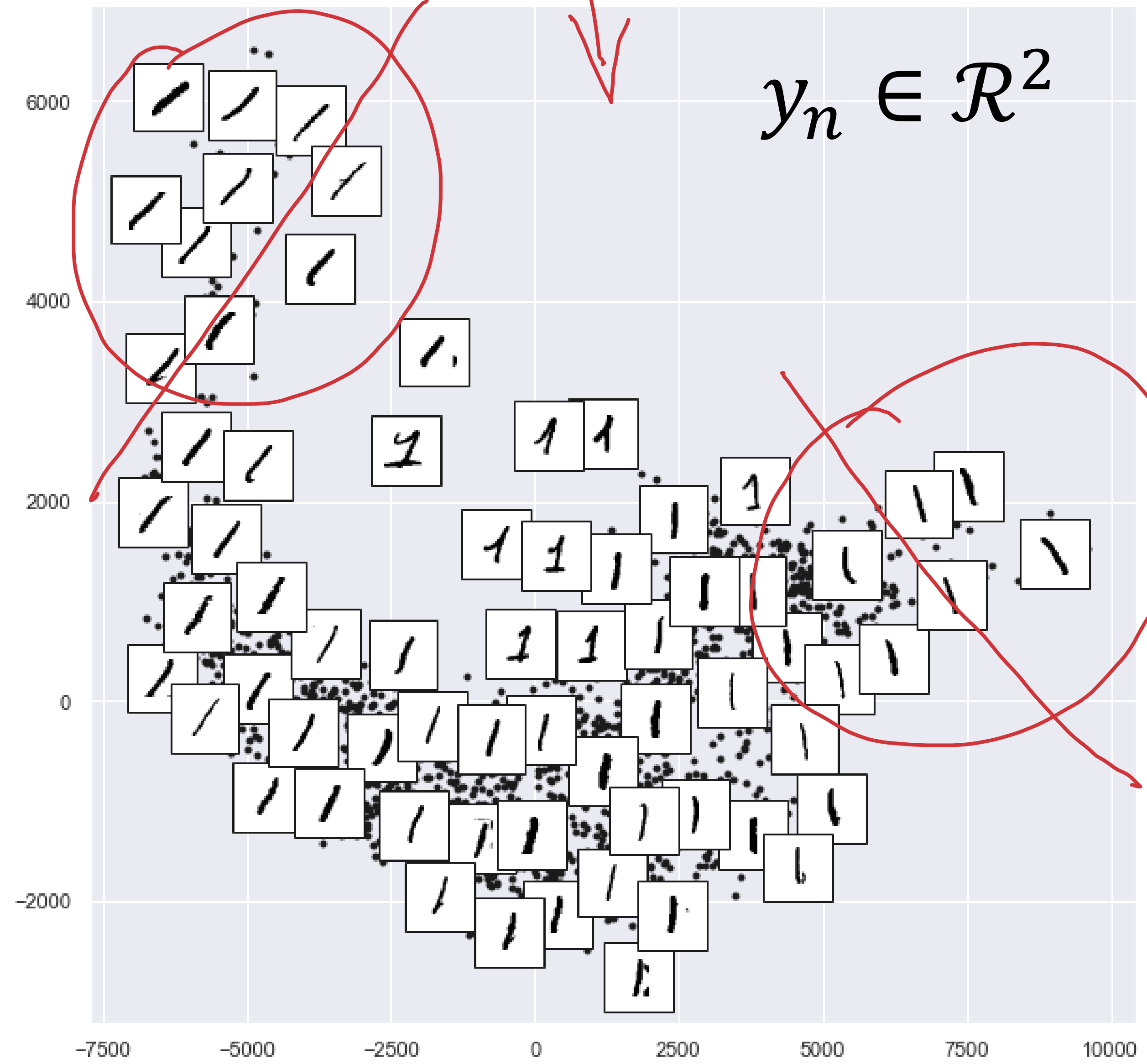
28x28

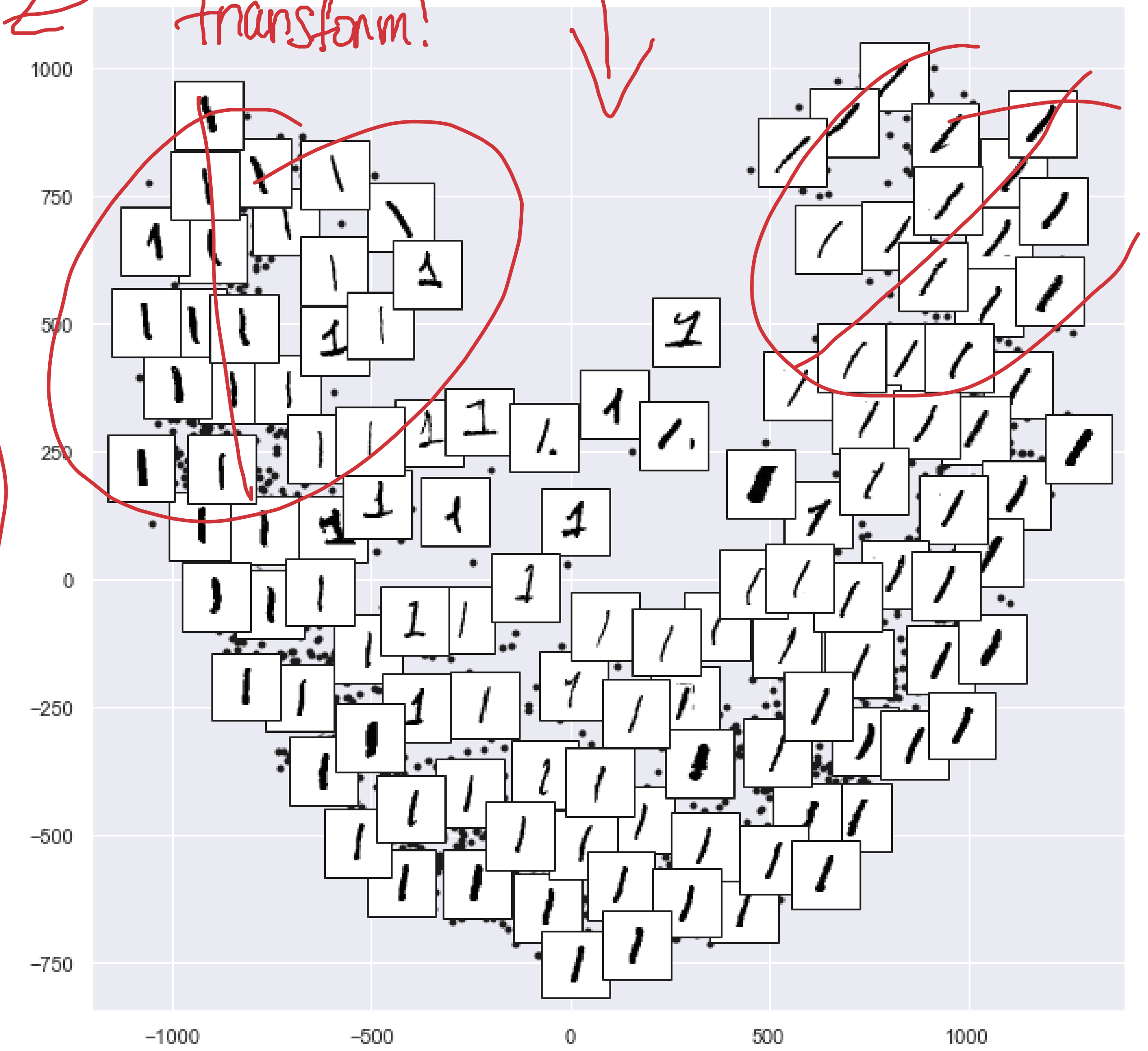


$x_n \in \mathcal{R}^{784}$

often transform!

$y_n \in \mathcal{R}^2$





Spectral Embedding

- Input: N data points $\{x_1, x_2, x_3, \dots, x_N\}$ and $x_n \in \mathcal{R}^M$
- We can build a neighbor graph of the data points
- Objective: to find N data points $\{y_1, y_2, y_3, \dots, y_N\}$ and $y_n \in \mathcal{R}^K$ such that the loss function is minimized:

$$\sum_{i,j} \boxed{w_{i,j}} \|y_i - y_j\|_2^2$$

$$\begin{cases} w_{12} = 1 \\ \|y_1 - y_2\|_2^2 \approx 0 \end{cases} \quad | \; y_1 = x_1$$

where $w_{i,j}$ is the similarity between x_i and x_j

So, it means if x_i and x_j are similar to each other, then y_i and y_j should be similar to each other, which means the distance between y_i and y_j should be very small.

Spectral Embedding

- Input: N data points $\{x_1, x_2, x_3, \ldots, x_N\}$ and $x_n \in \mathcal{R}^M$
- **Step-1**: build a neighbor graph of the data points
- **Step-2**: compute the so-called graph Laplacian $L = D - A$
- **Step-3**: compute K *smallest* eigenvalues and corresponding eigenvectors of L

  the eigenvectors are denoted by $v_1, v_2, \ldots, v_K$

  and put them into a matrix $V = [v_1, v_2, \ldots, v_K]$, a N-by-K matrix

Output:

y_n is the n-th row of V

Spectral Embedding

- Input: N data points $\{x_1, x_2, x_3, \ldots, x_N\}$ and $x_n \in \mathcal{R}^M$
- **Step-1**: build a neighbor graph of the data points
- **Step-2**: compute the so-called graph Laplacian $L = D - A$
- **Step-3**: compute K *smallest* eigenvalues and corresponding eigenvectors of L

  the eigenvectors are denoted by $v_1, v_2, \ldots, v_K$

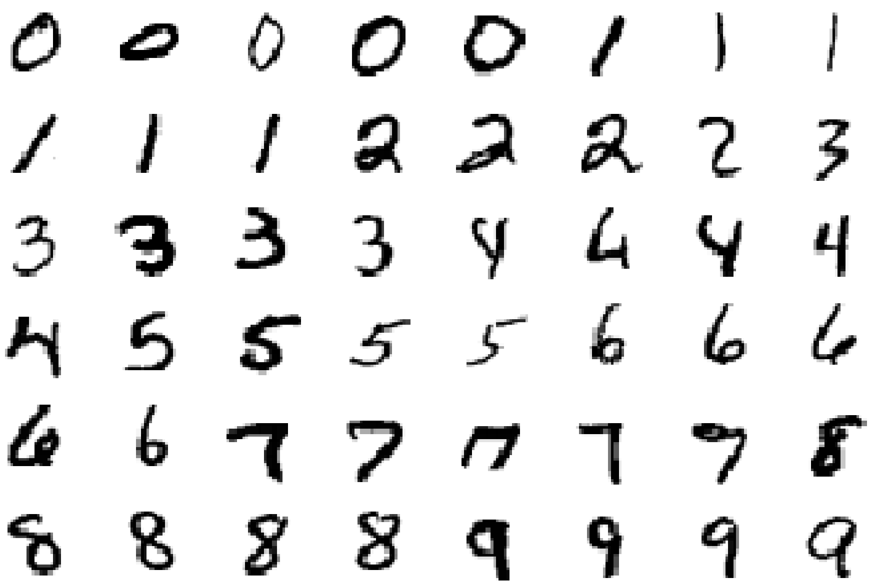
  and put them into a matrix $V = [v_1, v_2, \ldots, v_K]$, a N-by-K matrix

Output: y_n is the n-th row of V

Spectral Clustering

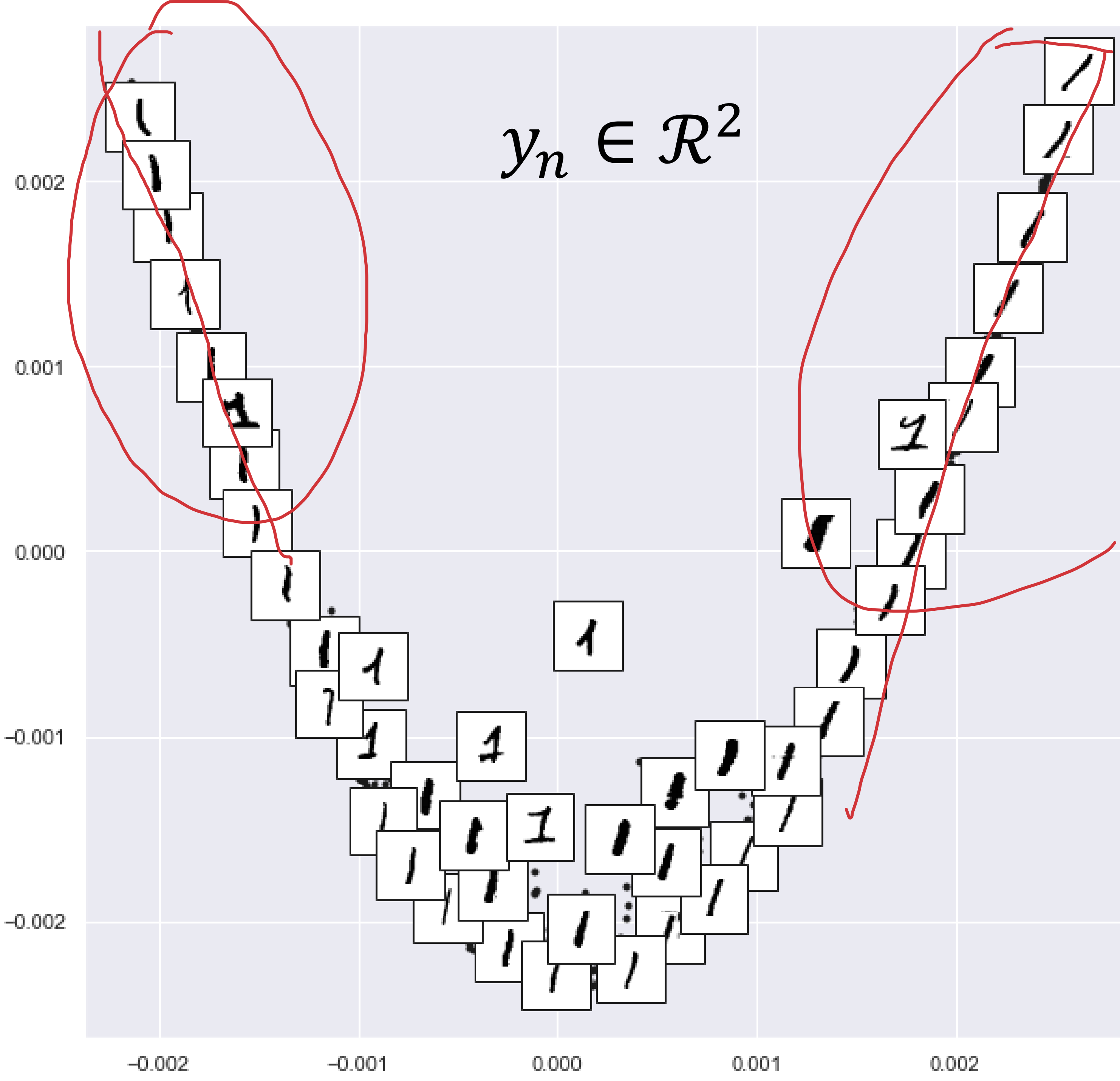
run k-means algorithm on output data $\{y_1, y_2, y_3, \ldots, y_N\}$

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.8
0.6
0.4
0.2
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
SE
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
−0.0075 −0.0050 −0.0025 0.0000 0.0025 0.0050 0.0075



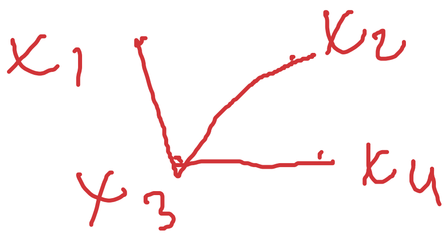
$x_n \in \mathcal{R}^{784}$

SE



t-Distributed Stochastic Neighbor Embedding (t-SNE)

- https://lvdmaaten.github.io/tsne/ (PnD)



$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)},$$

similarity of data point x_j to x_i

probability distribution about the distribution

$q_{1|i} + q_{2|i} + q_{3|i} = 1$

$$q_{j|i} \triangleq \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}.$$

similarity of data point y_j to y_i

$\|P_i - Q_i\|_2^2$

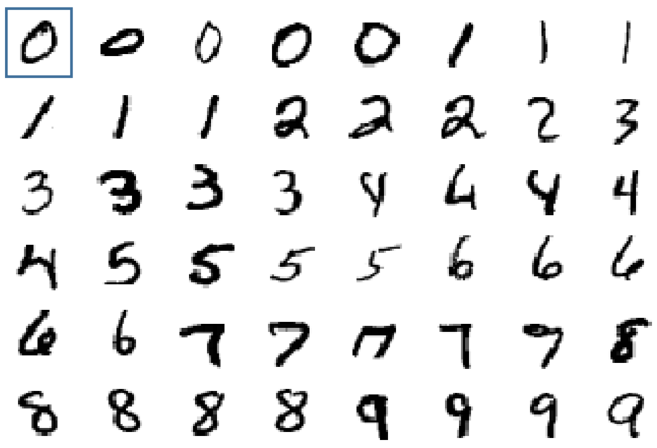
minimize Kullback–Leibler divergence:

the "average distance" between the two sets of similarities/prob-distributions

$$C = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}},$$

- $P_i = \{P_{1|i}, P_{2|i}, P_{3|i}\}$
- $Q_i = \{Q_{1|i}, Q_{2|i}, Q_{3|i}\}$

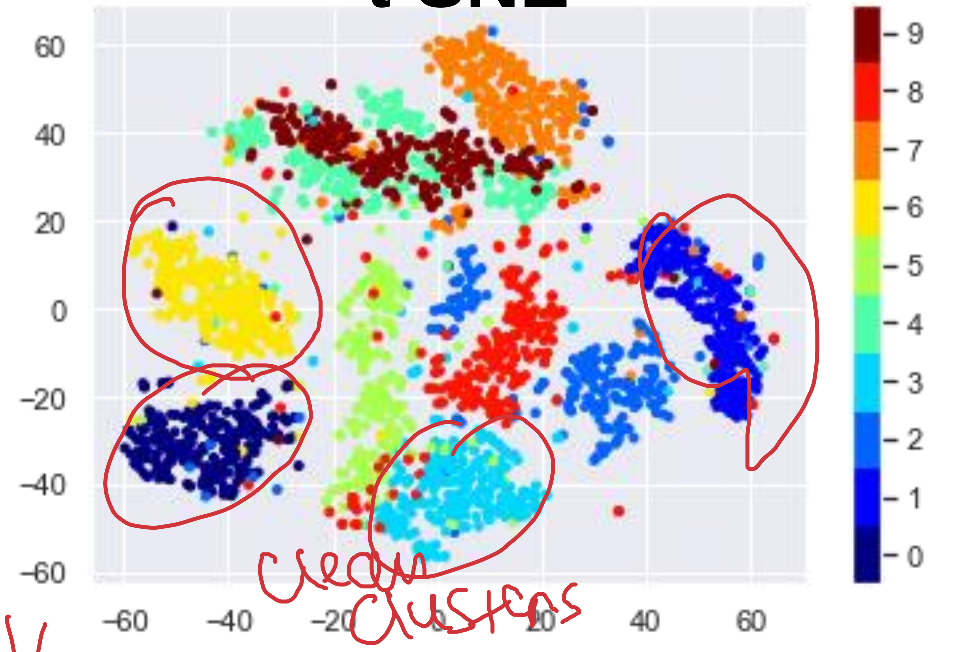
$x_n \in \mathcal{R}^{784}$



Converted: reduce dimention: In 2D

M

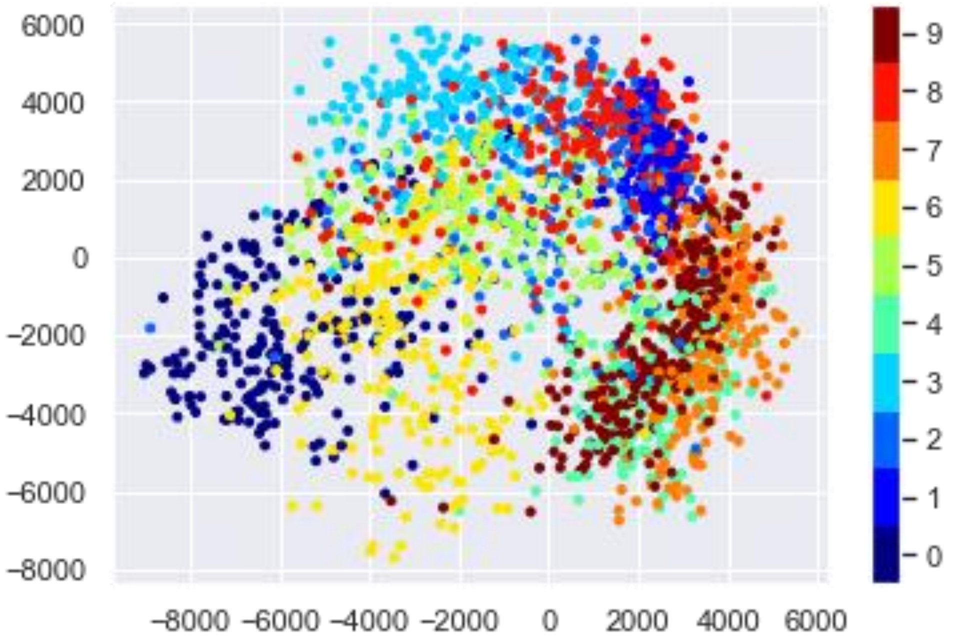
t-SNE



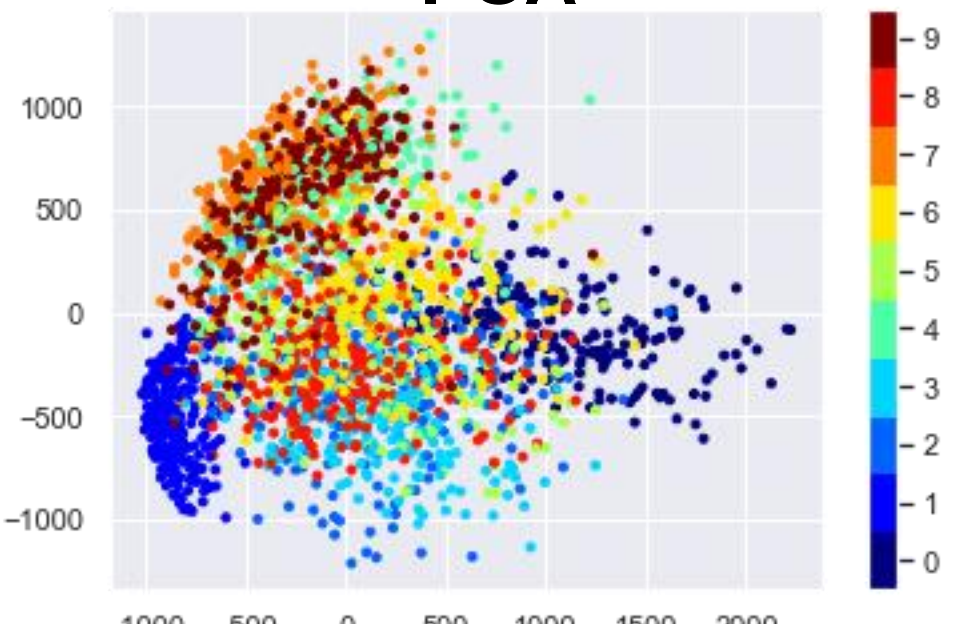
$y_n \in \mathcal{R}^2$

K

isomap



PCA



Why dimensionality reduction ? (Why?)

- The dimension-reduced data can be used for
  - Visualizing, exploring and understanding the data
  - Cleaning the data (assuming data = information +noise)
  - Speeding up subsequent learning task
  - Building simpler models

speed up the process!