

# DSC 140B

*Representation Learning*

Lecture 15 | Part 1

**Solving the Optimization Problem**

# A New Formulation

- ▶ **Given:** an  $n \times n$  similarity matrix  $W$
- ▶ **Compute:** embedding vector  $\vec{f}$  **minimizing**

$$\text{Cost}(\vec{f}) = \vec{f}^T L \vec{f}$$

subject to  $\|\vec{f}\| = 1$  and  $\vec{f} \perp (1, 1, \dots, 1)^T$

- ▶ This might sound familiar...

# Recall: PCA

- ▶ **Given:** a  $d \times d$  covariance matrix  $C$
- ▶ **Find:** vector  $\vec{u}$  **maximizing** the variance in the direction of  $\vec{u}$ :

$$\vec{u}^T C \vec{u}$$

subject to  $\|\vec{u}\| = 1$ .

- ▶ **Solution:** take  $\vec{u}$  = top eigenvector of  $C$

# A New Formulation

- ▶ Forget about orthogonality constraint for now.
- ▶ **Compute:** embedding vector  $\vec{f}$  **minimizing**

$$\text{Cost}(\vec{f}) = \vec{f}^T L \vec{f}$$

subject to  $\|\vec{f}\| = 1$ .

- ▶ **Solution:** the *bottom* eigenvector of  $L$ .
  - ▶ That is, eigenvector with smallest eigenvalue.

# Claim

- ▶ The bottom eigenvector is  $\vec{f} = \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$
- ▶ It has associated eigenvalue of 0.
- ▶ That is,  $L\vec{f} = 0\vec{f} = \vec{0}$

# Spectral<sup>1</sup> Theorem

## Theorem

*If  $A$  is a symmetric matrix, eigenvectors of  $A$  with distinct eigenvalues are orthogonal to one another.*

---

<sup>1</sup>“Spectral” not in the sense of specters (ghosts), but because the eigenvalues of a transformation form the “spectrum”

# The Fix

- ▶ Remember: we wanted  $\vec{f}$  to be orthogonal to  $\frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$ .
  - ▶ i.e., should be orthogonal to bottom eigenvector of  $L$ .
- ▶ Fix: take  $\vec{f}$  to be eigenvector of  $L$  with with smallest eigenvalue  $\neq 0$ .
- ▶ Will be  $\perp \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$  by the **spectral theorem**.

# Spectral Embeddings: Problem

- ▶ **Given:** **similarity graph** with  $n$  nodes
- ▶ **Compute:** an **embedding** of the  $n$  points into  $\mathbb{R}^1$  so that similar objects are placed nearby
- ▶ **Formally:** find embedding vector  $\vec{f}$  **minimizing**

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (f_i - f_j)^2 = \vec{f}^T L \vec{f}$$

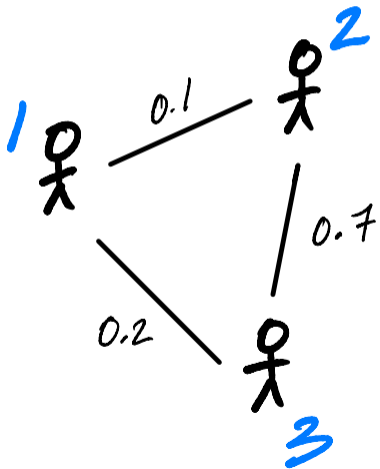
subject to  $\|\vec{f}\| = 1$  and  $\vec{f} \perp (1, 1, \dots, 1)^T$



# Spectral Embeddings: Solution

- ▶ Form the **graph Laplacian** matrix,  $L = D - W$
- ▶ Choose  $\vec{f}$  be an eigenvector of  $L$  with smallest eigenvalue  $> 0$
- ▶ This is the embedding!

# Example



```
W = np.array([
    [1, 0.1, 0.2],
    [0.1, 1, 0.7],
    [0.2, 0.7, 1]
])
```

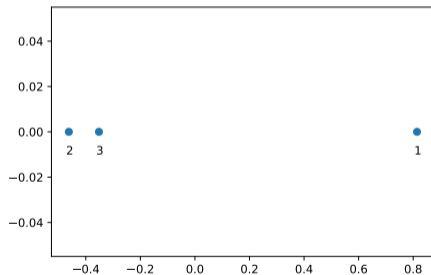
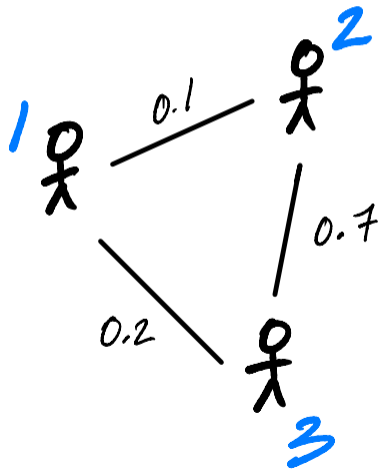
```
D = np.diag(W.sum(axis=1))
```

```
L = D - W
```

```
vals, vecs = np.linalg.eigh(L)
```

```
f = vecs[:,1]
```

# Example



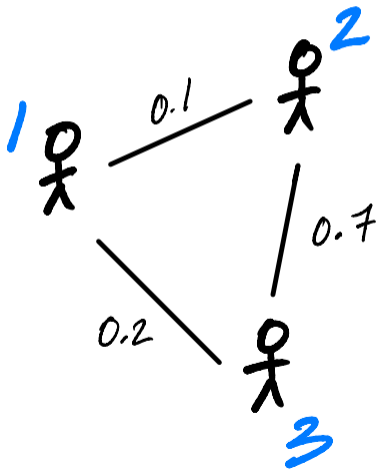
# Embedding into $\mathbb{R}^k$

- ▶ This embeds nodes into  $\mathbb{R}^1$ .
- ▶ What about embedding into  $\mathbb{R}^k$ ?
- ▶ Natural extension: find bottom  $k$  eigenvectors with eigenvalues  $> 0$

# New Coordinates

- ▶ With  $k$  eigenvectors  $\vec{f}^{(1)}, \vec{f}^{(2)}, \dots, \vec{f}^{(k)}$ , each node is mapped to a point in  $\mathbb{R}^k$ .
- ▶ Consider node  $i$ .
  - ▶ First new coordinate is  $f_i^{(1)}$ .
  - ▶ Second new coordinate is  $f_i^{(2)}$ .
  - ▶ Third new coordinate is  $f_i^{(3)}$ .
  - ▶  $\vdots$

# Example



```
W = np.array([
    [1, 0.1, 0.2],
    [0.1, 1, 0.7],
    [0.2, 0.7, 1]
])
```

```
D = np.diag(W.sum(axis=1))
L = D - W
```

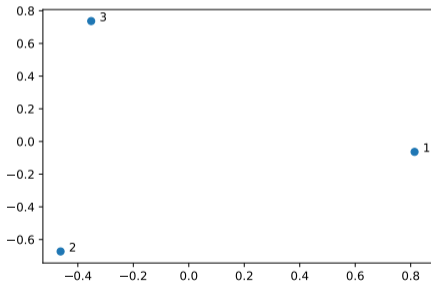
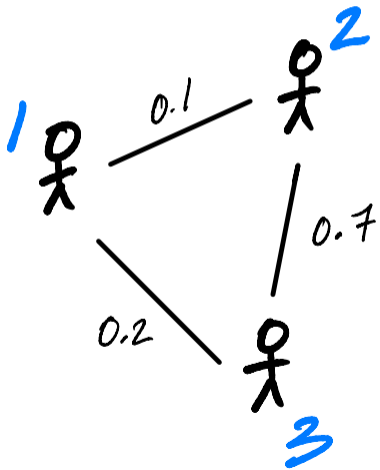
```
vals, vecs = np.linalg.eigh(L)
```

```
# take two eigenvectors
```

```
# to map to  $R^2$ 
```

```
f = vecs[:,1:3]
```

# Example



# Laplacian Eigenmaps

- ▶ This approach is part of the method of “**Laplacian eigenmaps**”
- ▶ Introduced by Mikhail Belkin<sup>2</sup> and Partha Niyogi
- ▶ It is a type of **spectral embedding**

---

<sup>2</sup>Now at HDSI



# A Practical Issue

- ▶ The Laplacian is often **normalized**:

$$L_{\text{norm}} = D^{-1/2} L D^{-1/2}$$

where  $D^{-1/2}$  is the diagonal matrix whose  $i$ th diagonal entry is  $1/\sqrt{d_{ii}}$ .

- ▶ Proceed by finding the eigenvectors of  $L_{\text{norm}}$ .

# In Summary

- ▶ We can **embed** a similarity graph's nodes into  $\mathbb{R}^k$  using the eigenvectors of the graph Laplacian
- ▶ Yet another instance where eigenvectors are solution to optimization problem
- ▶ Next time: using this for dimensionality reduction

# DSC 140B

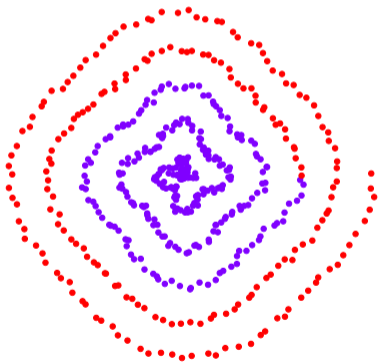
*Representation Learning*

Lecture 15 | Part 2

**Nonlinear Dimensionality Reduction**

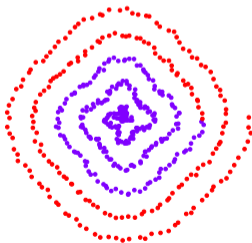
# Scenario

- ▶ You want to train a classifier on this data.
- ▶ It would be easier if we could “unroll” the spiral.
- ▶ Data seems to be one-dimensional, even though in two dimensions.
- ▶ Dimensionality reduction?



# PCA?

- ▶ Does PCA work here?
- ▶ Try projecting onto one principal component.



**No**



# PCA?

- ▶ PCA simply “rotates” the data.
- ▶ No amount of rotation will “unroll” the spiral.
- ▶ We need a fundamentally different approach that works for non-linear patterns.

# Today

- ▶ Non-linear dimensionality reduction via **spectral embeddings**.

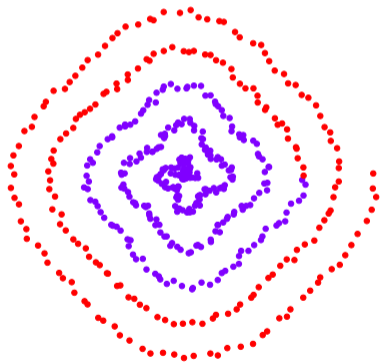


# Last Time: Spectral Embeddings

- ▶ **Given:** a similarity graph with  $n$  nodes, number of dimensions  $k$ .
- ▶ **Embed:** each node as a point in  $\mathbb{R}^k$  such that similar nodes are mapped to nearby points
- ▶ **Solution:** *bottom*  $k$  non-constant eigenvectors of graph Laplacian

# Idea

- ▶ Build a similarity graph from points.
- ▶ Points *near the spiral* should be similar.
- ▶ Embed the similarity graph into  $\mathbb{R}^1$



# Today

- ▶ 1) How do we build a graph from a set of points?
- ▶ 2) Dimensionality reduction with Laplacian eigenmaps

# DSC 140B

## Representation Learning

Lecture 15 | Part 3

**From Points to Graphs**

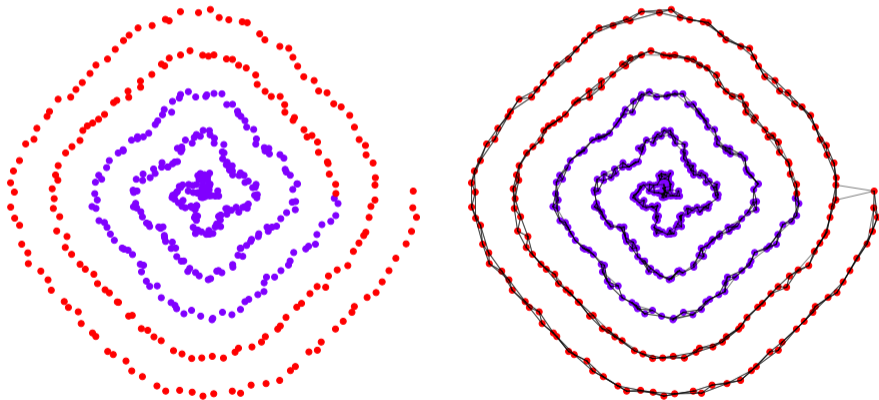
# Dimensionality Reduction

- ▶ **Given:**  $n$  points in  $\mathbb{R}^d$ , number of dimensions  $k \leq d$
- ▶ **Map:** each point  $\vec{x}$  to new representation  $\vec{z} \in \mathbb{R}^k$

# Idea

- ▶ Build a similarity graph from points in  $\mathbb{R}^2$
- ▶ Use approach from last lecture to embed into  $\mathbb{R}^k$
- ▶ But how do we represent a set of points as a similarity graph?

# Why graphs?



# Three Approaches

- ▶ 1) Epsilon neighbors graph
- ▶ 2)  $k$ -Nearest neighbor graph
- ▶ 3) fully connected graph with similarity function



# Epsilon Neighbors Graph

- ▶ Input: vectors  $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$ , a number  $\epsilon$
- ▶ Create a graph with one node  $i$  per point  $\vec{x}^{(i)}$
- ▶ Add edge between nodes  $i$  and  $j$  if  $\|\vec{x}^{(i)} - \vec{x}^{(j)}\| \leq \epsilon$
- ▶ Result: **unweighted** graph

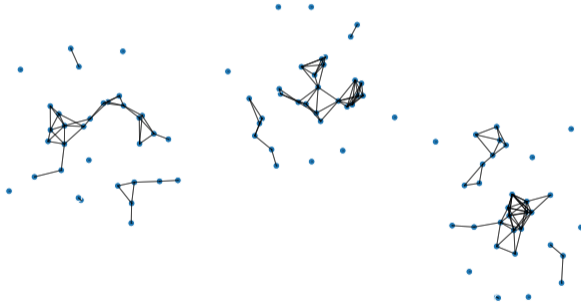


## Exercise

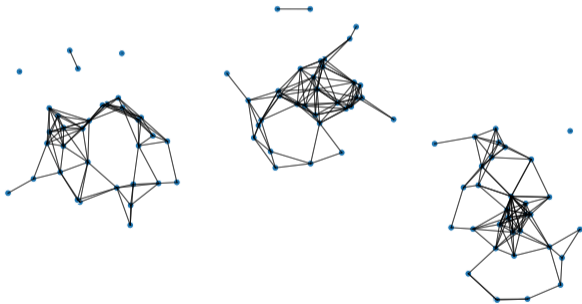
What will the graph look like when  $\epsilon$  is small? What about when it is large?



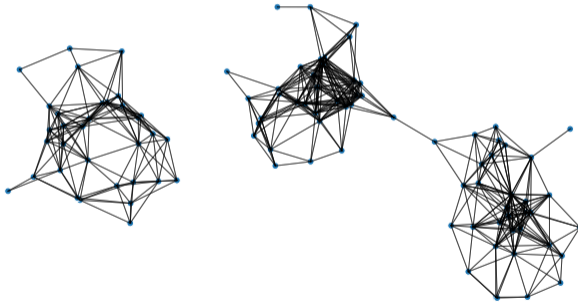
# Epsilon Neighbors Graph



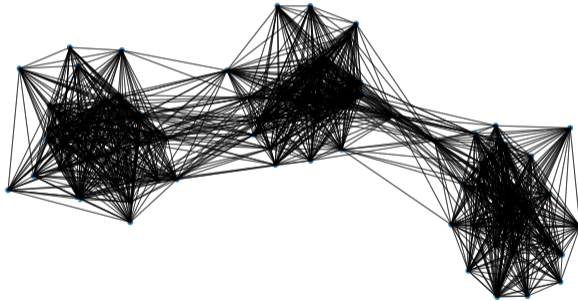
# Epsilon Neighbors Graph



# Epsilon Neighbors Graph



# Epsilon Neighbors Graph



## Note

- ▶ We've drawn these graphs by placing nodes at the same position as the point they represent
- ▶ But a graph's nodes can be drawn in any way

# Epsilon Neighbors: Pseudocode

```
# assume the data is in X
n = len(X)
adj = np.zeros_like(X)
for i in range(n):
    for j in range(n):
        if distance(X[i], X[j]) <= epsilon:
            adj[i, j] = 1
```



## Picking $\varepsilon$

- ▶ If  $\varepsilon$  is too small, graph is underconnected
- ▶ If  $\varepsilon$  is too large, graph is overconnected
- ▶ If you cannot visualize, just try and see

## With scikit-learn

```
import sklearn.neighbors
adj = sklearn.neighbors.radius_neighbors_graph(
    X,
    radius=...
)
```

# k-Neighbors Graph

- ▶ Input: vectors  $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$ , a number  $k$
- ▶ Create a graph with one node  $i$  per point  $\vec{x}^{(i)}$
- ▶ Add edge between each node  $i$  and its  $k$  closest neighbors
- ▶ Result: **unweighted** graph



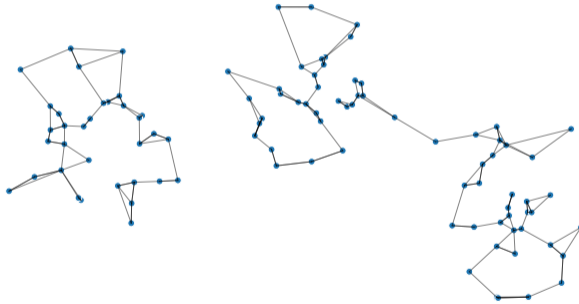
# k-Neighbors: Pseudocode

```
# assume the data is in X  
n = len(X)  
adj = np.zeros_like(X)  
for i in range(n):  
    for j in k_closest_neighbors(X, i):  
        adj[i, j] = 1
```

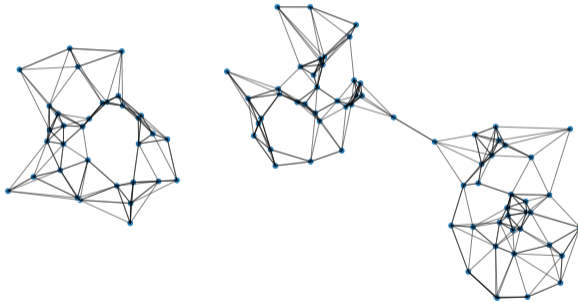
## Exercise

Is it possible for a  $k$ -neighbors graph to be disconnected?

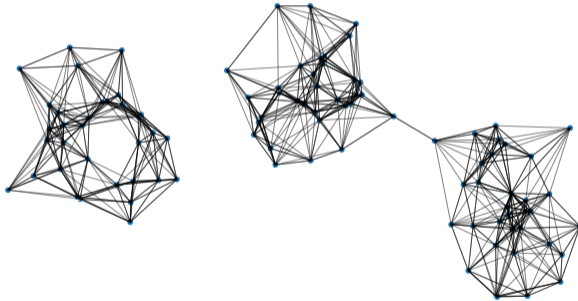
# k-Neighbors Graph



# k-Neighbors Graph

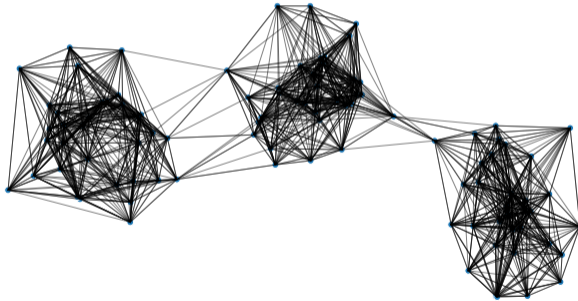


# k-Neighbors Graph





# k-Neighbors Graph



## With scikit-learn

```
import sklearn.neighbors
adj = sklearn.neighbors.kneighbors_graph(
    X,
    n_neighbors=...
)
```

# Fully Connected Graph

- ▶ Input: vectors  $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$ , a similarity function  $h$
- ▶ Create a graph with one node  $i$  per point  $\vec{x}^{(i)}$
- ▶ Add edge between every pair of nodes. Assign weight of  $h(\vec{x}^{(i)}, \vec{x}^{(j)})$
- ▶ Result: **weighted** graph



# Gaussian Similarity

- ▶ A common similarity function: Gaussian
- ▶ Must choose  $\sigma$  appropriately

$$h(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|^2/\sigma^2}$$

# Fully Connected: Pseudocode

```
def h(x, y):  
    dist = np.linalg.norm(x, y)  
    return np.exp(-dist**2 / sigma**2)  
  
# assume the data is in X  
n = len(X)  
w = np.ones_like(X)  
for i in range(n):  
    for j in range(n):  
        w[i, j] = h(X[i], X[j])
```

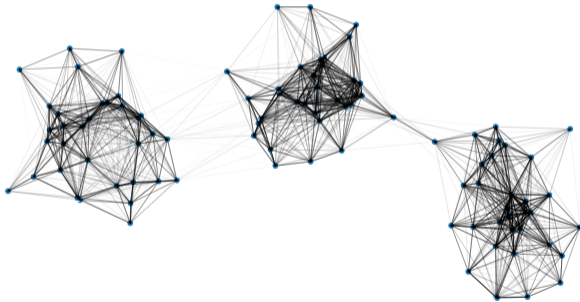
## With SciPy

```
distances = scipy.spatial.distance_matrix(X, X)
w = np.exp(-distances**2 / sigma**2)
```

# Gaussian Similarity

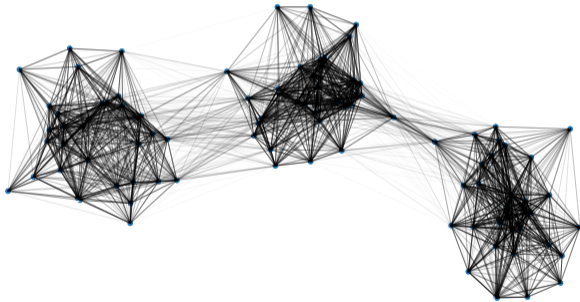


# Gaussian Similarity

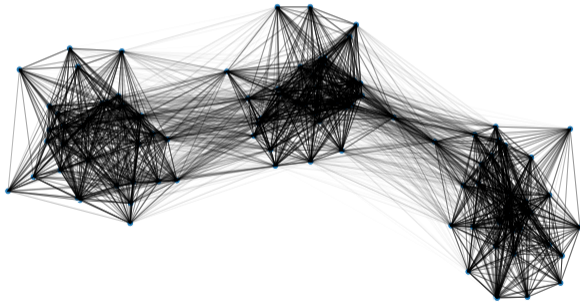




# Gaussian Similarity



# Gaussian Similarity



# DSC 140B

## Representation Learning

Lecture 15 | Part 4

**Laplacian Eigenmaps**

# Problem: Graph Embedding

- ▶ **Given:** a similarity graph, target dimension  $k$
- ▶ **Goal:** **embed** the nodes of the graph as points in  $\mathbb{R}^k$  so that similar nodes are nearby
- ▶ **(One) Solution:** Embed using eigenvectors of the graph Laplacian

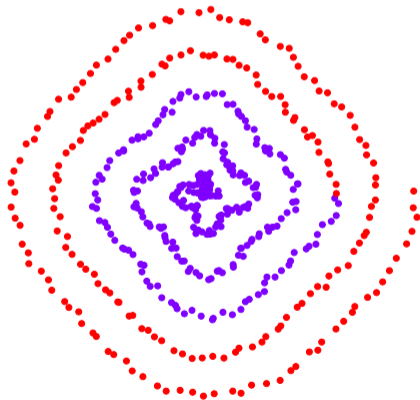
# Problem: Non-linear Dimensionality Reduction

- ▶ **Given:** points in  $\mathbb{R}^d$ , target dimension  $k$
- ▶ **Goal:** **embed** the points in  $\mathbb{R}^k$  so that points that were close in  $\mathbb{R}^d$  are close after

# Idea

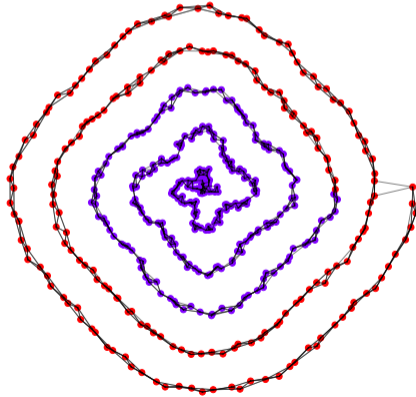
- ▶ Build a similarity graph from points in  $\mathbb{R}^d$ 
  - ▶ epsilon neighbors,  $k$ -neighbors, or fully connected
- ▶ Embed the similarity graph in  $\mathbb{R}^k$  using eigenvectors of graph Laplacian

# Example 1: Spiral



# Example 1: Spiral

- ▶ Build a  $k$ -neighbors graph.
- ▶ Note: follows the 1-d shape of the data.





# Example 1: Spectral Embedding

- ▶ Let  $W$  be the weight matrix ( $k$ -neighbor adjacency matrix)
- ▶ Compute  $L = D - W$
- ▶ Compute bottom  $k$  non-constant eigenvectors of  $L$ , use as embedding

# Example 1: Spiral

- ▶ Embedding into  $\mathbb{R}^1$



# Example 1: Spiral

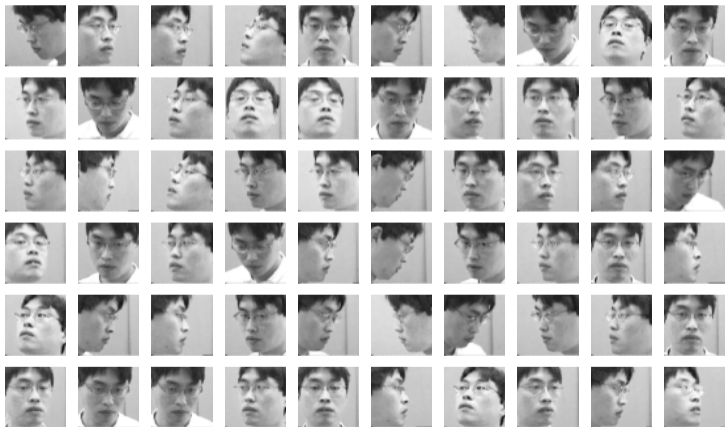
- ▶ Embedding into  $\mathbb{R}^2$



# Example 1: Spiral

```
import sklearn.neighbors
import sklearn.manifold
W = sklearn.neighbors.kneighbors_graph(
    X, n_neighbors=4
)
embedding = sklearn.manifold.spectral_embedding(
    W, n_components=2
)
```

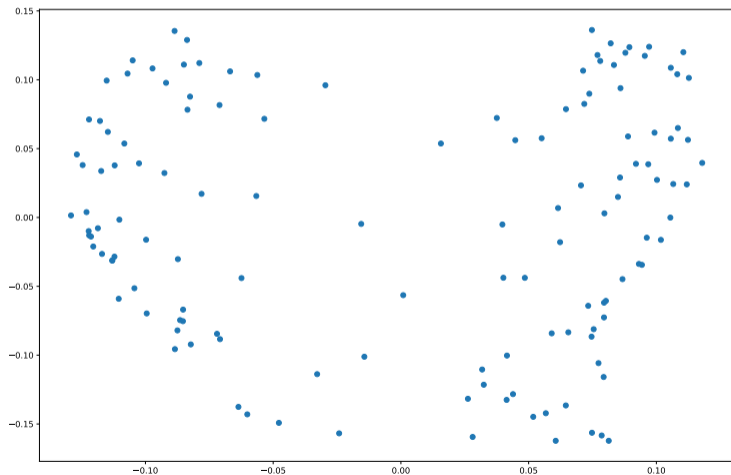
## Example 2: Face Pose



## Example 2: Face Pose

- ▶ Construct fully-connected similarity graph with Gaussian similarity
- ▶ Embed with Laplacian eigenmaps

# Example 2: Face Pose



# Example 2: Face Pose

