Change of Basis

- Let $\mathcal{U} = \{\hat{u}^{(1)}, \ldots, \hat{u}^{(d)}\}$ be an orthonormal basis.
- The coordinates of $\vec{x}$ w.r.t. $\mathcal{U}$ are:

$$[\vec{x}]_{\mathcal{U}} = \begin{pmatrix} \vec{x} \cdot \hat{u}^{(1)} \\ \vec{x} \cdot \hat{u}^{(2)} \\ \vdots \\ \vec{x} \cdot \hat{u}^{(d)} \end{pmatrix}$$

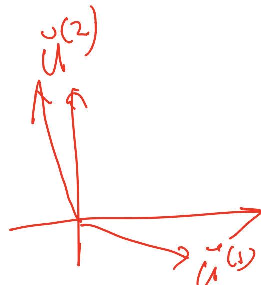
Exercise

Let $\vec{x} = (-1, 4)^T$ and suppose:

$$\hat{u}^{(1)} \cdot \hat{e}^{(1)} = 3 \qquad \hat{u}^{(2)} \cdot \hat{e}^{(1)} = -1$$

$$\hat{u}^{(1)} \cdot \hat{e}^{(2)} = -2 \qquad \hat{u}^{(2)} \cdot \hat{e}^{(2)} = 5$$

What is $[\vec{x}]_{\mathcal{U}}$?



Recall: Linear Transformations

- A **transformation** $\vec{f}(\vec{x})$ is a function which takes a vector as input and returns a vector of the same dimensionality.

- A transformation $\vec{f}$ is **linear** if

$$\vec{f}(\alpha\vec{u} + \beta\vec{v}) = \alpha\vec{f}(\vec{u}) + \beta\vec{f}(\vec{v})$$

Implications of Linearity

- Suppose $\vec{f}$ is a linear transformation. Then:

$$\begin{aligned}\vec{f}(\vec{x}) &= \vec{f}(x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)}) \\ &= x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)})\end{aligned}$$

- I.e., $\vec{f}$ is **totally determined** by what it does to the basis vectors.

Eigenvectors

- Let A be an $n \times n$ matrix. An **eigenvector** of A with **eigenvalue** λ is a nonzero vector $\vec{v}$ such that $A\vec{v} = \lambda\vec{v}$.

Variance in a Direction

- Let $\vec{u}$ be a unit vector.
- $z^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$ is the new feature for $\vec{x}^{(i)}$.
- The variance of the new features is:

$$\begin{aligned}
\text{Var}(z) &= \frac{1}{n} \sum_{i=1}^{n} (z^{(i)} - \mu_z)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\vec{x}^{(i)} \cdot \vec{u} - \mu_z\right)^2
\end{aligned}$$

Note

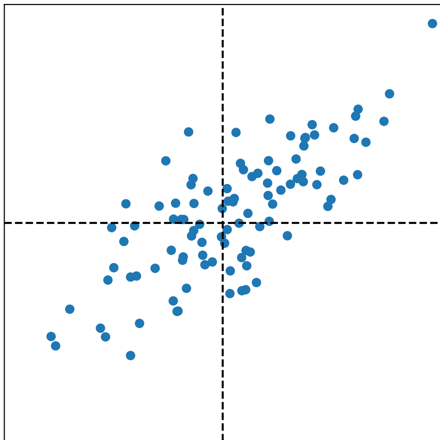
- If the data are centered, then $\mu_z = 0$ and the variance of the new features is:

$$
\begin{aligned}
\text{Var}(z) &= \frac{1}{n} \sum_{i=1}^{n} (z^{(i)})^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\vec{x}^{(i)} \cdot \vec{u}\right)^2
\end{aligned}
$$

Claim

$$\frac{1}{n}\sum_{i=1}^{n}\left(\vec{x}^{(i)}\cdot\vec{u}\right)^2 = \vec{u}^T C \vec{u}$$

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} & \\ & \end{pmatrix}$$

PCA: k Components

- Given data $\{\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}\} \in \mathbb{R}^d$, number of components k.
- Compute covariance matrix C, top $k \leq d$ eigenvectors $\vec{u}^{(1)}$, $\vec{u}^{(2)}$, ..., $\vec{u}^{(k)}$.
- For any vector $\vec{x} \in \mathbb{R}$, its new representation in $\mathbb{R}^k$ is $\vec{z} = (z_1, z_2, \ldots z_k)^T$, where:

$$
\begin{aligned}
z_1 &= \vec{x} \cdot \vec{u}^{(1)} \\
z_2 &= \vec{x} \cdot \vec{u}^{(2)} \\
&\vdots \\
z_k &= \vec{x} \cdot \vec{u}^{(k)}
\end{aligned}
$$

Reconstructions

- Given a "new" representation of $\vec{x}$, $\vec{z} = (z_1, \ldots, z_k) \in \mathbb{R}^k$
- And top k eigenvectors, $\vec{u}^{(1)}, \ldots, \vec{u}^{(k)}$
- The **reconstruction** of $\vec{x}$ is

$$z_1 \vec{u}^{(1)} + z_2 \vec{u}^{(2)} + \ldots + z_k \vec{u}^{(k)} = U\vec{z}$$

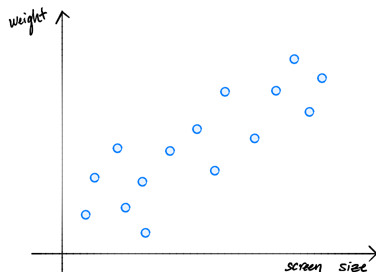
Reconstruction Error

- The reconstruction *approximates* the original point, $\vec{x}$.
- The **reconstruction error** for a single point, $\vec{x}$:

$$\|\vec{x} - U\vec{z}\|^2$$

- Total reconstruction error:

$$\sum_{i=1}^{n} \|\vec{x}^{(i)} - U\vec{z}^{(i)}\|^2$$



Total Variance

- The **total variance** is the sum of the eigenvalues of the covariance matrix.
- Or, alternatively, sum of variances in each orthogonal basis direction.

DSC 140B
Representation Learning

Lecture 15 | Part 1

The Graph Laplacian

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 $\vec{f} \perp (1, 1, \ldots, 1)^T$

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!

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 $\vec{f}^{(1)}, \vec{f}^{(2)}, \ldots, \vec{f}^{(k)}$, each node is mapped to a point in $\mathbb{R}^k$.

- Consider node i.
  - First new coordinate is $\vec{f}_i^{(1)}$.
  - Second new coordinate is $\vec{f}_i^{(2)}$.
  - Third new coordinate is $\vec{f}_i^{(3)}$.
  - $\vdots$

Laplacian Eigenmaps

- This approach is part of the method of "**Laplacian eigenmaps**"
- Introduced by Mikhail Belkin[3] and Partha Niyogi
- It is a type of **spectral embedding**

[3] 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 ith diagonal entry is $1/\sqrt{d_{ii}}$.

- Proceed by finding the eigenvectors of L_{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

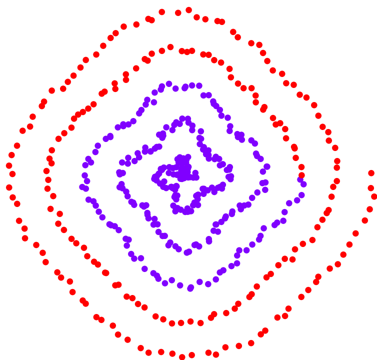
Representation Learning

Lecture 15 | Part 2

Nonlinear Dimensionality Reduction

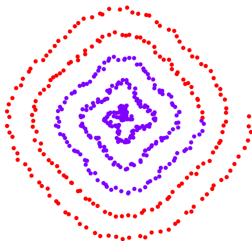
Scenario

- 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

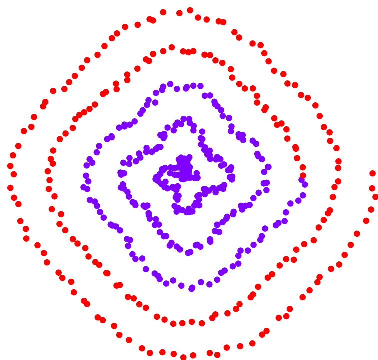
- Non-linear dimensionality reduction via **spectral embeddings**.

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

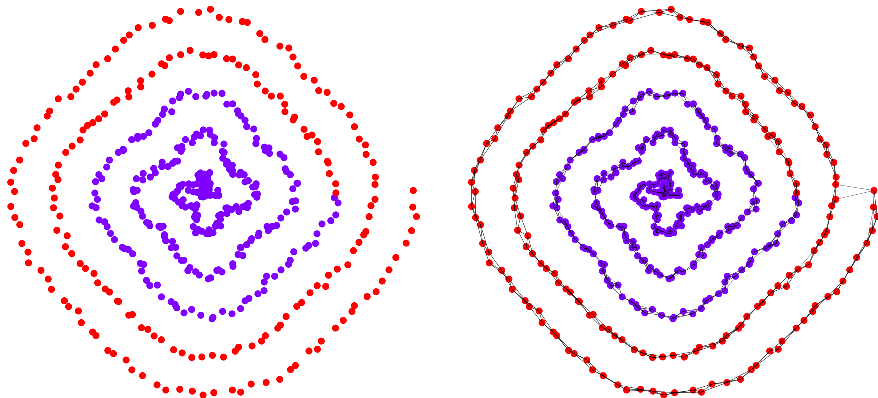
- **Given**: n points in $\mathbb{R}^d$, number of dimensions $k \leq d$

- **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

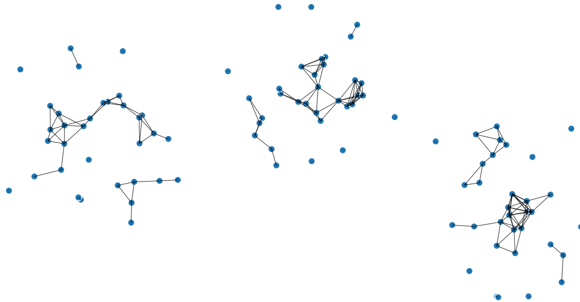
Epsilon Neighbors Graph

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

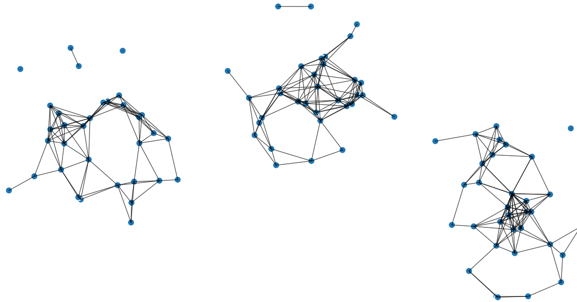
Exercise

What will the graph look like when ε is small? What about when it is large?

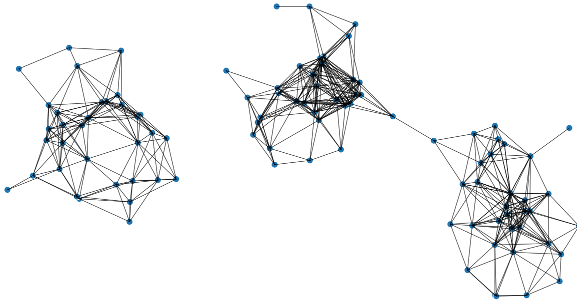
Epsilon Neighbors Graph



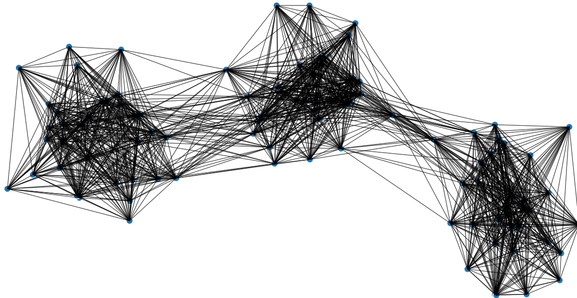
Epsilon Neighbors Graph



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 ε

- If ε is too small, graph is underconnected
- If ε 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 $\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}$, 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

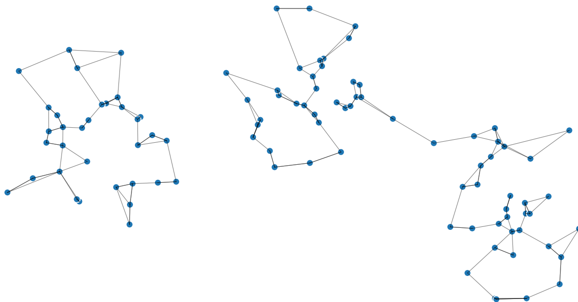
k-Neighbors: Pseudocode

```
# 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
```

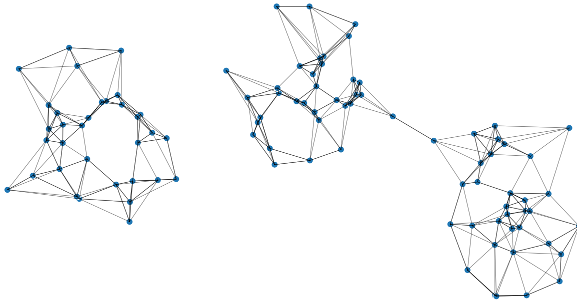
Exercise

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

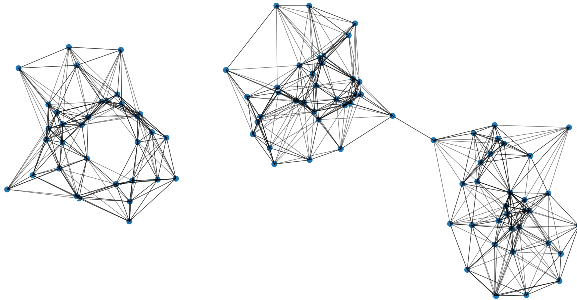
k-Neighbors Graph



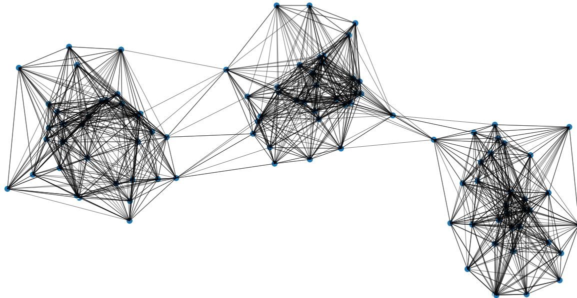
k-Neighbors Graph



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 σ appropriately

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

Fully Connected: Pseudocode

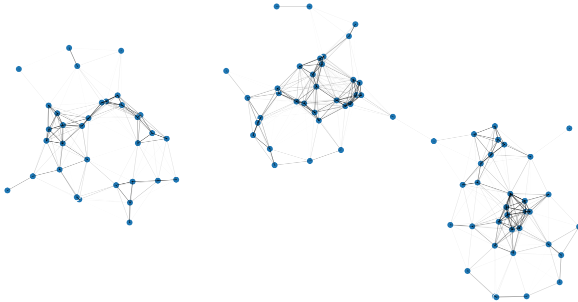
```
def h(x, y):
    dist = np.linalg.norm(x, y)
    return np.exp(-dist**2 / sigma**2)

# 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])
```

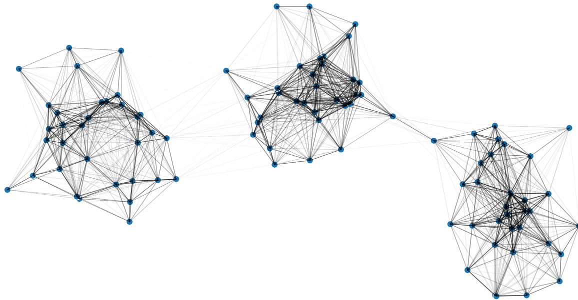
With SciPy

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

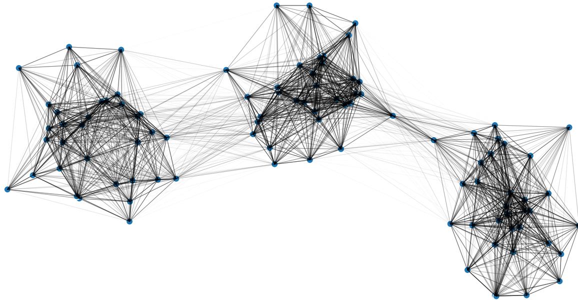
Gaussian Similarity



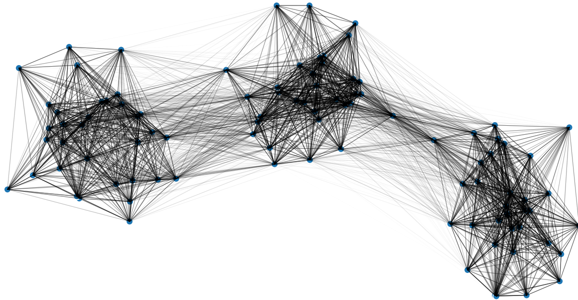
Gaussian Similarity



Gaussian Similarity



Gaussian Similarity



DSC 140B
Representation Learning

Lecture 15 | Part 4

Summary: 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

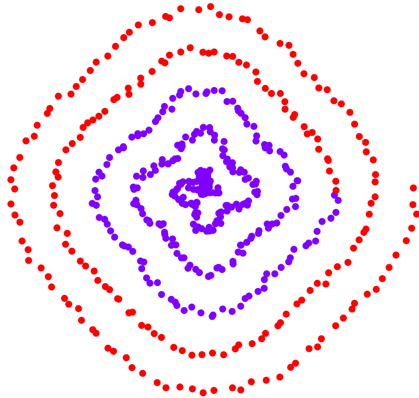
Problem: Non-linear Dimensionality Reduction

- **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

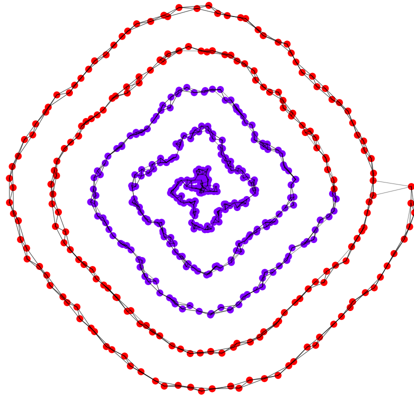
- Build a similarity graph from points in $\mathbb{R}^d$
  - epsilon neighbors, k-neighbors, or fully connected
- Embed the similarity graph in $\mathbb{R}^k$ using eigenvectors of graph Laplacian

Example 1: Spiral



Example 1: Spiral

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



Example 1: Spectral Embedding

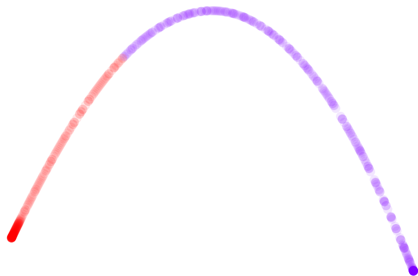
- Let W be the weight matrix (k-neighbor adjacency matrix)
- Compute $L = D - W$
- Compute bottom k non-constant eigenvectors of L, use as embedding

Example 1: Spiral

- Embedding into $\mathbb{R}^1$

Example 1: Spiral

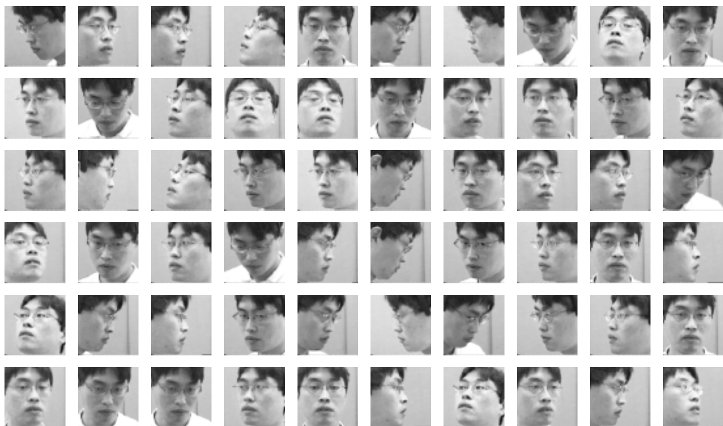
- Embedding into $\mathbb{R}^2$



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
)
```

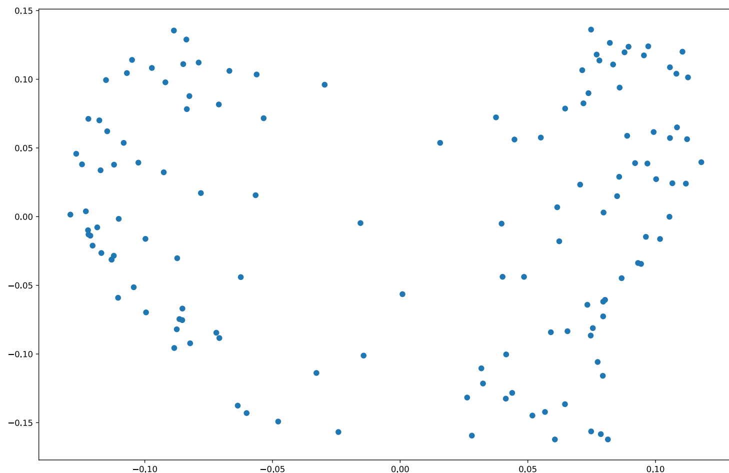
Example 2: Face Pose



Example 2: Face Pose

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

Example 2: Face Pose



Example 2: Face Pose