DSC 190

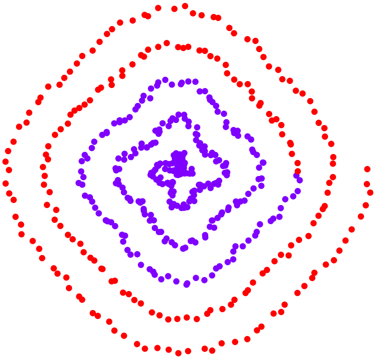
Machine Learning: Representations

Lecture 11 | Part 1

Nonlinear Dimensionality Reduction

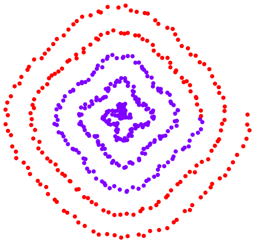
Scenario

- 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**.

$f(x)$ $\vec{f}_x$ $C(\vec{f}) = \sum_i \sum_j w_{ij}(f_i - f_j)^2$

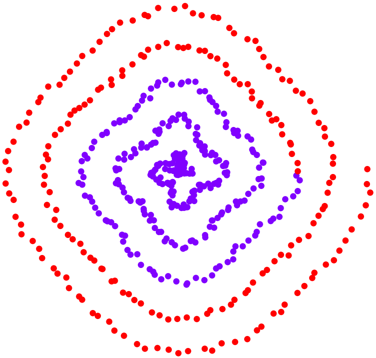
Last Time: Spectral Embeddings

$\vec{f}^T L \vec{f}$

- **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 190

Machine Learning: Representations

Lecture 11 | Part 2

From Points to Graphs

Dimensionality Reduction

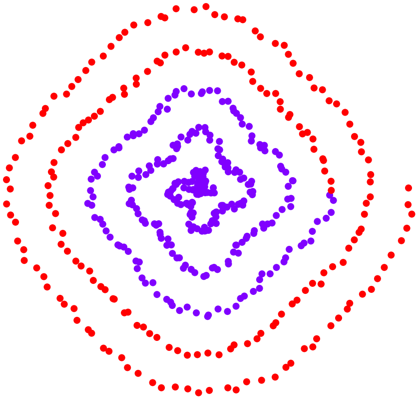
- **Given**: n points in $\mathbb{R}^d$, number of dimensions $k \leq d$

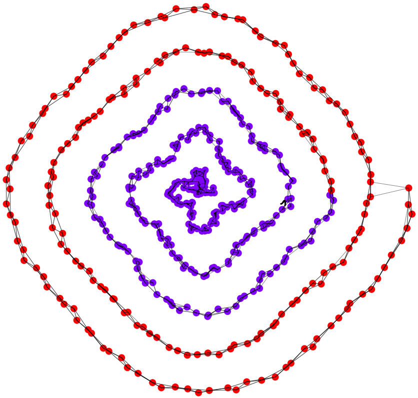
- **Map**: each point $\vec{x}$ to new representation $\vec{z} \in \mathbb{R}^k$

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

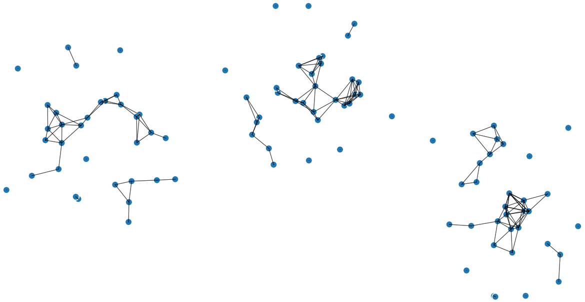
Epsilon Neighbors Graph

- 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

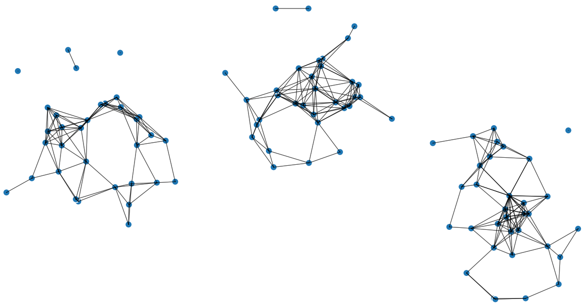
Exercise

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

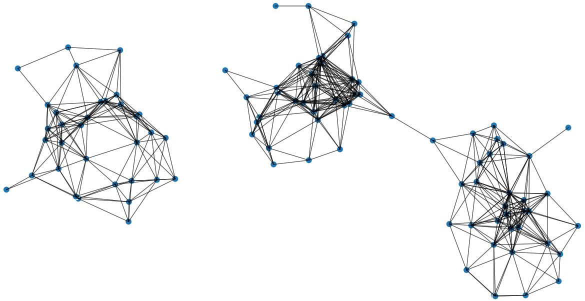
Epsilon Neighbors Graph



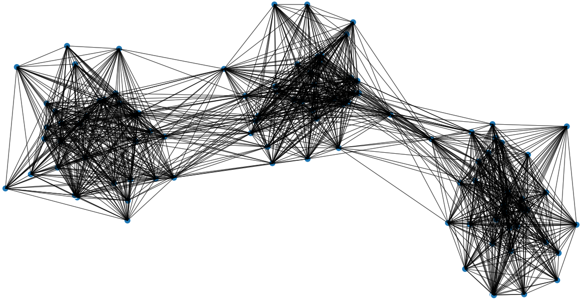
Epsilon Neighbors Graph



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) ((n, n))
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

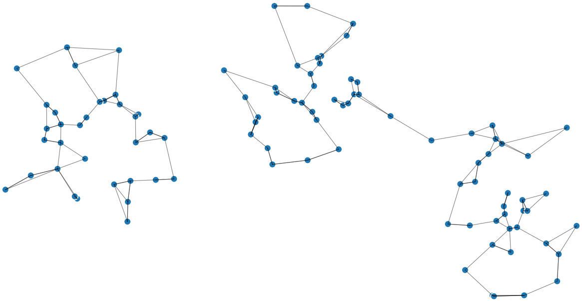
k-Neighbors: Pseudocode

```
# 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
```

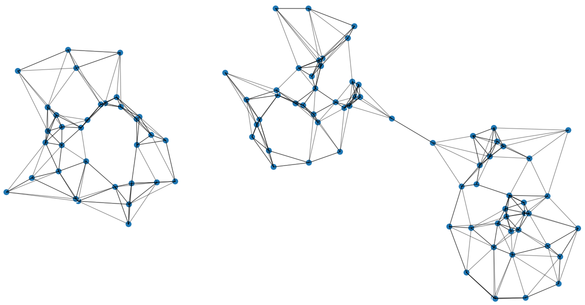
Exercise

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

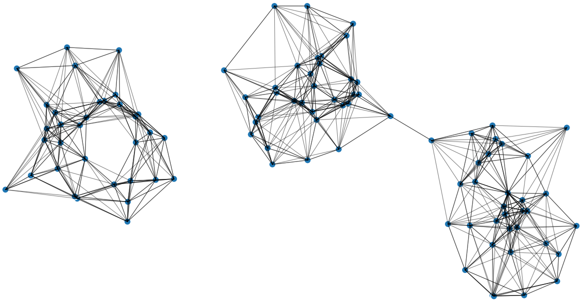
k-Neighbors Graph



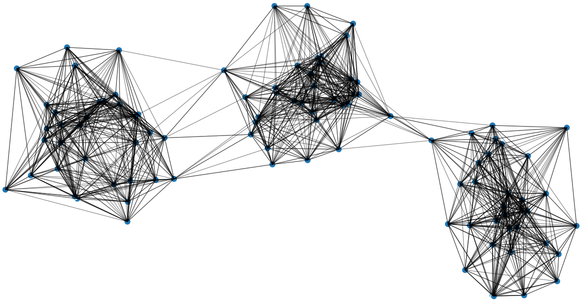
k-Neighbors Graph



k-Neighbors Graph



k-Neighbors Graph



With scikit-learn

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

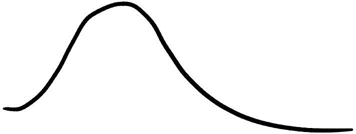
Fully Connected Graph

- 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

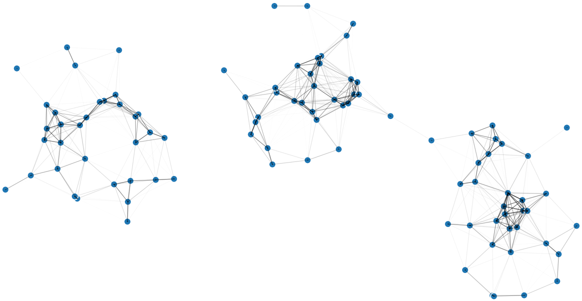
```python
def h(x, y):
    dist = np.linalg.norm(x, y)
    return np.exp(-dist**2 / sigma**2)

# 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])
```

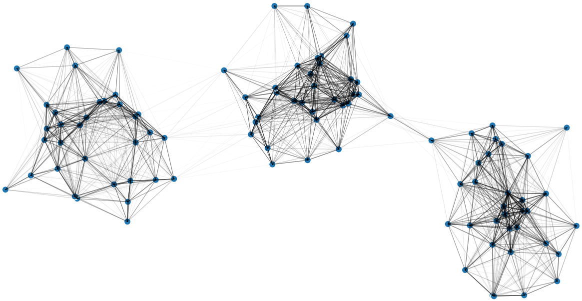
With SciPy

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

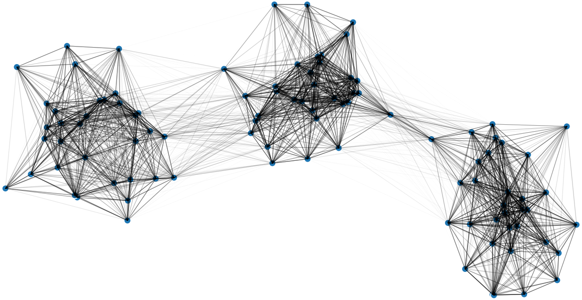
Gaussian Similarity



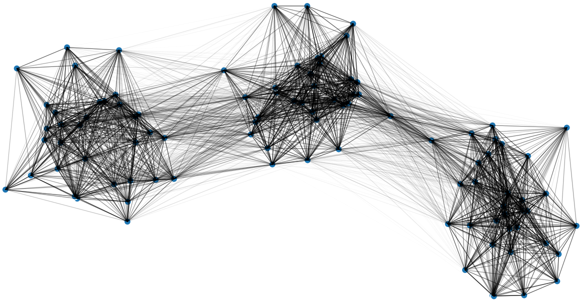
Gaussian Similarity



Gaussian Similarity



Gaussian Similarity



DSC 190

Machine Learning: Representations

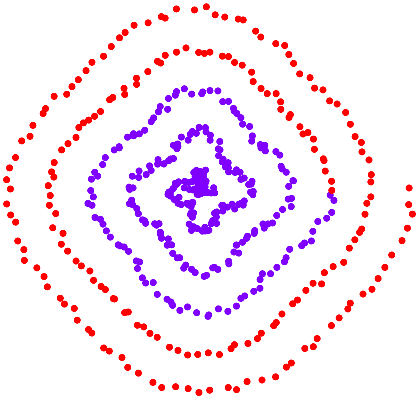
Lecture 11 | Part 3

Laplacian Eigenmaps

Idea

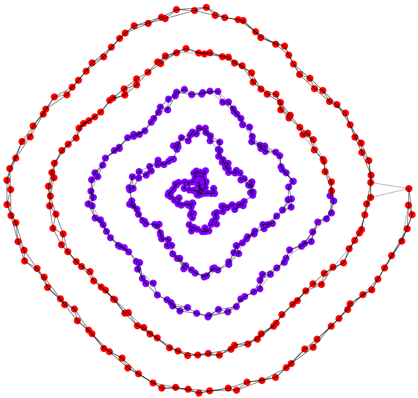
- Build a similarity graph from points in $\mathbb{R}^2$
  - epsilon neighbors, k-neighbors, or fully connected
- Now: use approach from last lecture to embed into $\mathbb{R}^k$

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 $L = D - W$

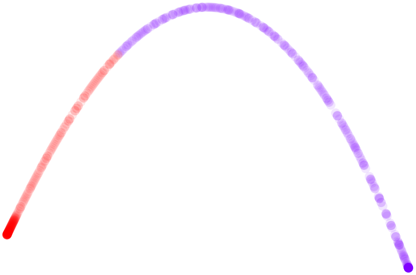
- Compute bottom k non-constant eigenvectors of L, use as embedding

Example 1: Spiral

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

Example 1: Spiral

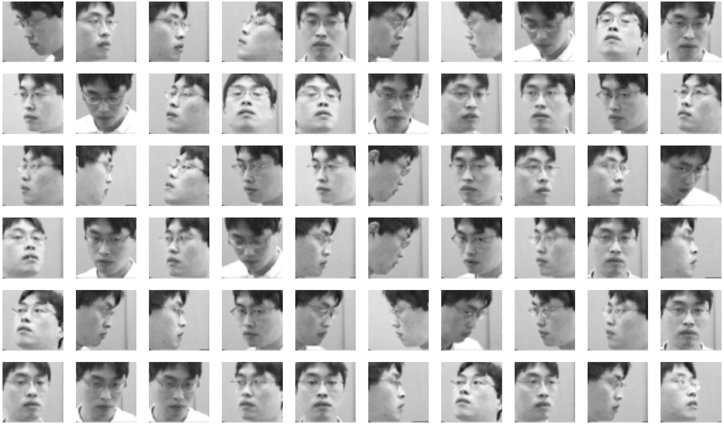
- Embedding into $\mathbb{R}^2$



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
)
```

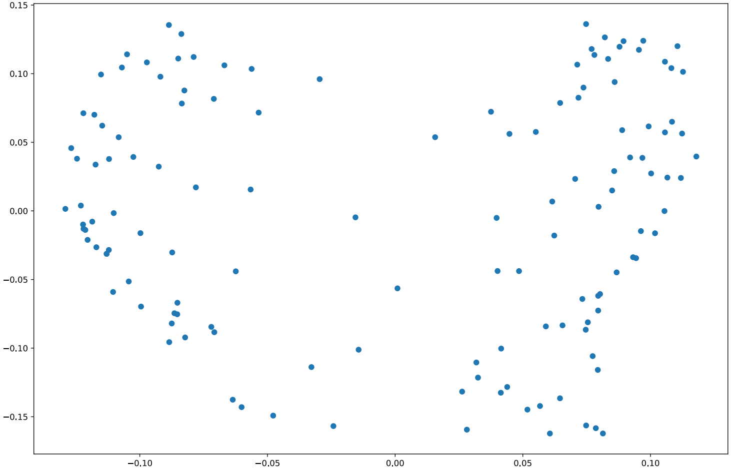
Example 2: Face Pose



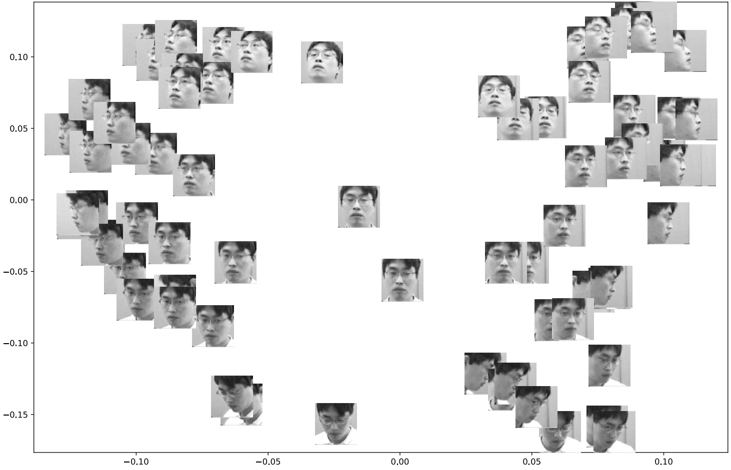
Example 2: Face Pose

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

Example 2: Face Pose



Example 2: Face Pose



DSC 190

Machine Learning: Representations

Lecture 11 | Part 4

Spectral Clustering

Spectral Embeddings

- Useful in multiple tasks:
  - Feature learning before classification
  - Visualizing high dimensional data
  - Clustering

Spectral Clustering

- Problem: k-means assumptions:
  - Data are vectors (what about graphs?)
  - Clusters are spherical (what about more complex patterns?)

- One idea:
  1. Embed using, e.g., Laplacian eigenmaps
  2. Run k-means on the embedded points

Demo