MATH 250: Mathematical Data Visualization

Manifold learning

Peter A. Gao

2024-03-25

San José State University

Outline

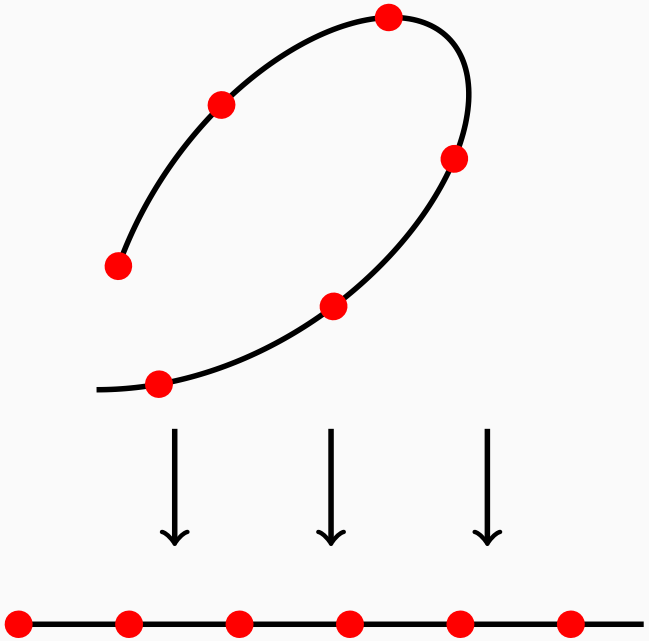
- Manifold learning and nonlinear embeddings
- ISOmap (adapted from lecture by Guangliang Chen)
- Previewing other approaches

Other readings

- Review of manifold learning by Meilă and Zhang (2024)
- Dimension reduction in R by Kraemer, Reichstein, and Mahecha (2018)
- Isomap paper by Tenenbaum, Silva, and Langford (2000)

Manifold learning

As we have discussed, linear embeddings (ex. MDS with Euclidean distances) can fail to represent non-linear geometries in data. Non-linear embedding methods yield strategies for better representing such data:



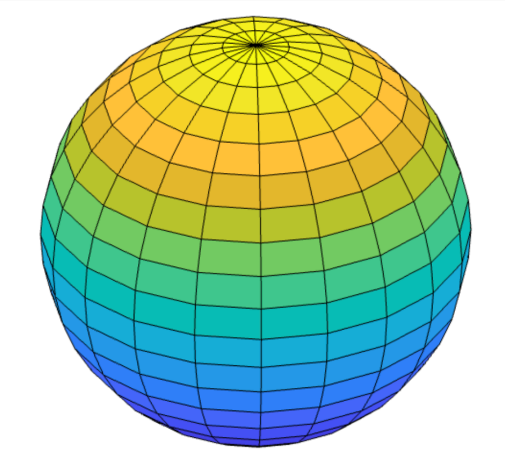
Manifold learning

Examples include:

- **Isometric feature mapping (Isomap)**
- **Locally linear embedding**
- **Laplacian eigenmaps**
- **t-SNE, UMAP**

What is a manifold?

Intuitively, a manifold is a space that resembles Euclidean space **locally**. For example, if you zoom in closely to any part of the sphere below, it resembles a two-dimensional plane.



What is a manifold?

Formal definition (Meilă and Zhang 2024):

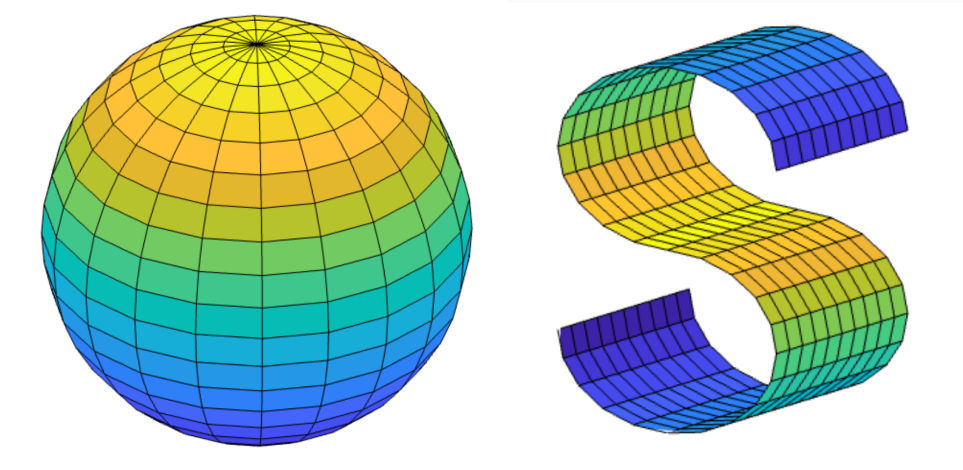
$\mathcal{M}$ is a smooth manifold of dimension q when it can be covered by patches U such that:

a. For each U, there is an invertible mapping $\varphi : U \to \mathbb{R}^q$ so that φ, φ^{-1} are smooth. Each pair (U, φ) is called a **chart** and $\mathbf{y} = \varphi(\mathbf{x})$ is the **local coordinate** of $\mathbf{x} \in \mathcal{M}$.

b. Whenever two charts (U, φ) and (V, ϕ) overlap, the change of coordinates $\varphi \circ \phi^{-1}$ is smooth on $\phi(U \cap V)$ and has a smooth inverse.

What is a manifold?

Examples of 2-manifolds:



Do data live on a manifold?

Linear dimensionality reduction techniques generally try to project high-dimensional data to a line or plane (or hyperplane).

Manifold learning techniques assume that high-dimensional data live approximately along a low-dimensional manifold. “Unwrapping” the manifold may help us visualize our high-dimensional data.

The manifold learning problem

Objective: Given a set of points in a high-dimensional Euclidean space along a q-manifold $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathcal{M} \subset \mathbb{R}^p$, where $\mathcal{M}$ is unknown, estimate $\mathcal{M}$ and derive a lower-dimensional representation $\mathbf{y}_1, \dots, \mathbf{y}_n \in \mathbb{R}^q$ (the local coordinates).

Manifold learning is also called nonlinear dimensionality reduction since manifolds are usually nonlinear.

The manifold assumption

Usually, we only observe data $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathcal{M} \subset \mathbb{R}^p$. We assume that our data are sampled from a distribution which has support on some q-dimensional manifold $\mathcal{M} \subset \mathbb{R}^p$.

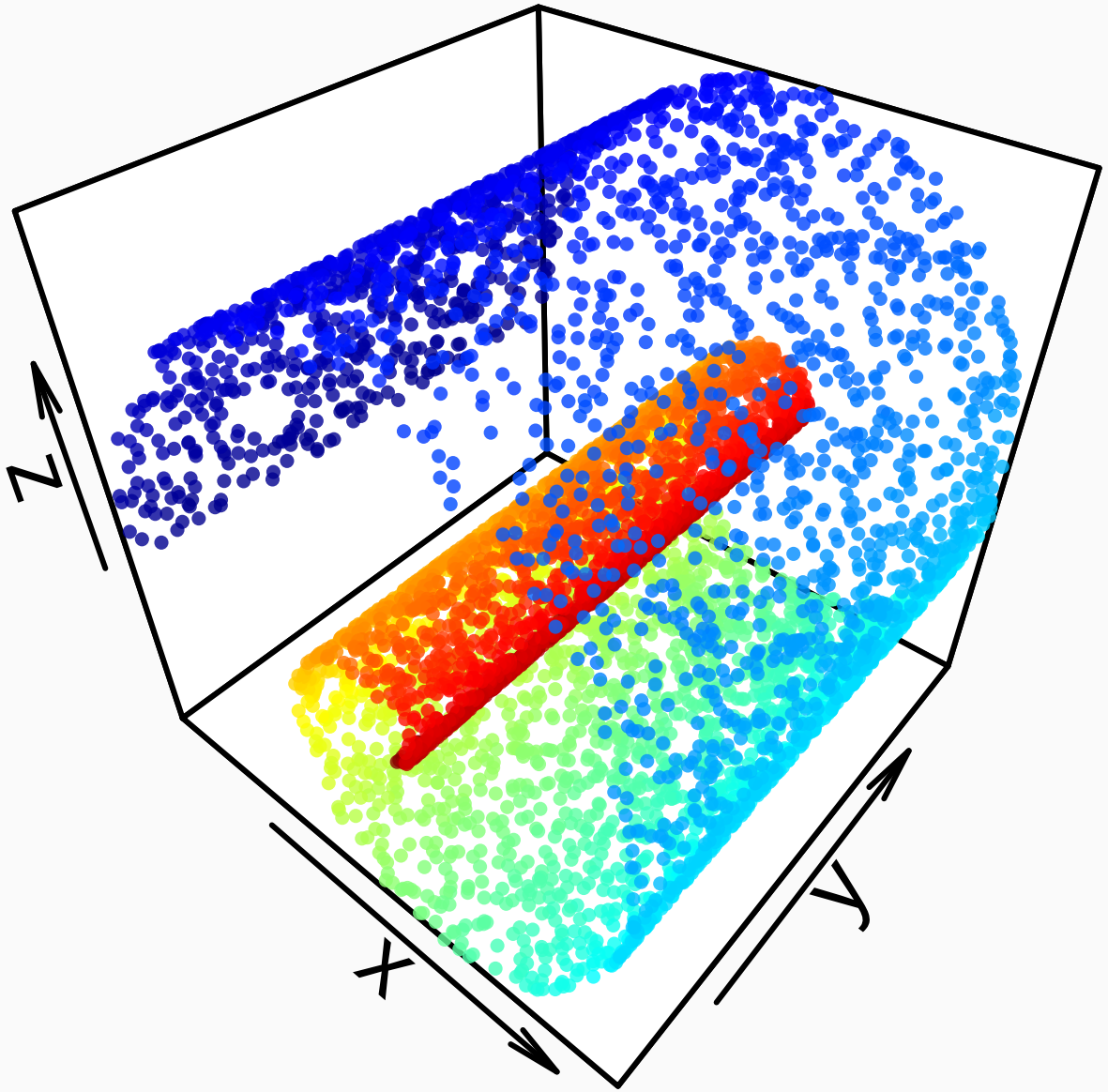
We often assume that the data lie on $\mathcal{M}$ (so there is no measurement error/noise).

Neighborhood graphs

Usually, manifold learning begins by finding each observation's "neighbors," which enables us to construct a neighborhood graph which summarizes the similarities/distances between observations. In essence, this provides a summary of the geometry of our observations

Neighborhood graph: a set of nodes and edges, where each data point $\mathbf{x}_i$ corresponds to a node, and an edge connects a pair of nodes if they are defined to be "neighbors."

Swiss roll data

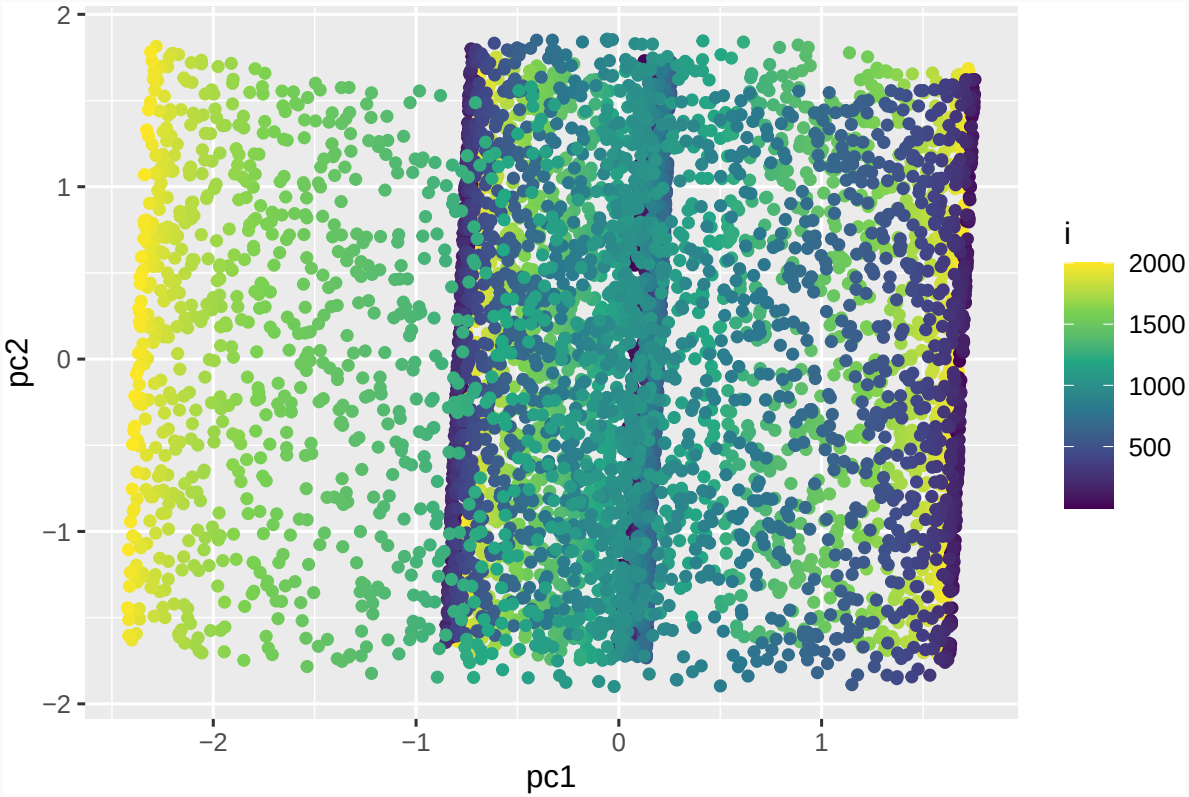


Swiss roll data

```
n <- 6000
theta <- seq(0, 3 * pi, length.out = n)
x <- theta * cos(theta)
z <- theta * sin(theta)
y <- runif(n)
library(plot3D)
scatter3D(x = x, y = y, z = z, pch = 16,
          alpha = .8, cex = .35,
          colvar = n:1,colkey = F, fill = T)
```

Swiss roll data: PCA

Applying PCA to this data yields the following principal components:



Swiss roll data: PCA code

```
data_mat <- cbind(x, y, z)
swissroll_pca <- prcomp(data_mat, scale = T)
rotated_dat <-
  data.frame(i = 1:2000,
             pc1 = swissroll_pca$x %*% swissroll_pca$rotation[, 1],
             pc2 = swissroll_pca$x %*% swissroll_pca$rotation[, 2])
library(ggplot2)
ggplot(rotated_dat, aes(x = pc1, y = pc2, color = i)) +
  geom_point() + scale_color_viridis_c()
```

Swiss roll data: PCA

In order to capture the nonlinear geometry, it is necessary to keep more dimensions (potentially more than the manifold dimension).

The principal directions may not be meaningful or interpretable if there is nonlinear geometry.

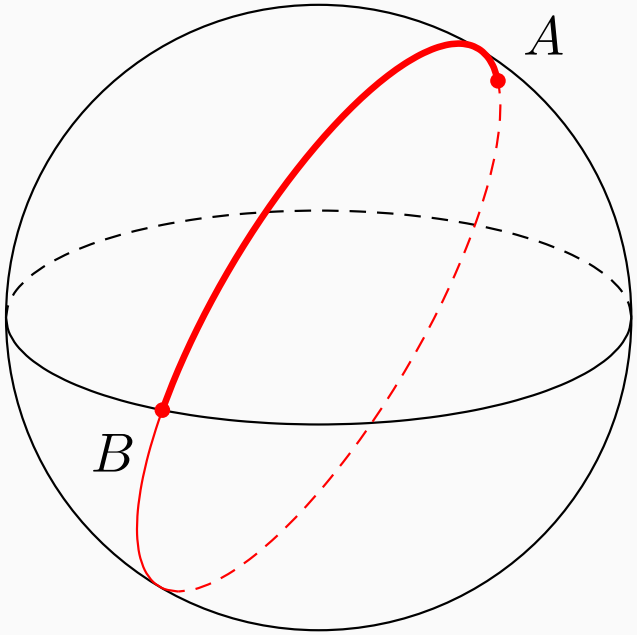
Isomap: MDS with geodesic distance

Recall that MDS tries to find a low-dimensional representation that preserves distances/dissimilarities between points in a high-dimensional space. By choosing a specific dissimilarity metric, we can achieve nonlinear dimensionality reduction.

The **geodesic distance** between two points $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{M}$ is denoted $d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathcal{M}}(i, j)$ and is defined as the shortest possible length of a path contained in $\mathcal{M}$ that connects $\mathbf{x}_i$ and $\mathbf{x}_j$.

Example: Great circle distance

On a sphere, the geodesic distance is the great circle distance.



Isomap: MDS with geodesic distance

In practice, we assume that we observe data $\mathbf{x}_1, \dots, \mathbf{x}_n$ sampled from a manifold $\mathcal{M}$, but we treat $\mathcal{M}$ itself as unknown.

We must approximate the true geodesic distances $d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathcal{M}}(i, j)$ somehow.

Idea: Build a nearest-neighbor graph G on the dataset and use the shortest-path distances on the graph to approximate $d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathcal{M}}(i, j)$.

Example: 2-D Swiss Roll

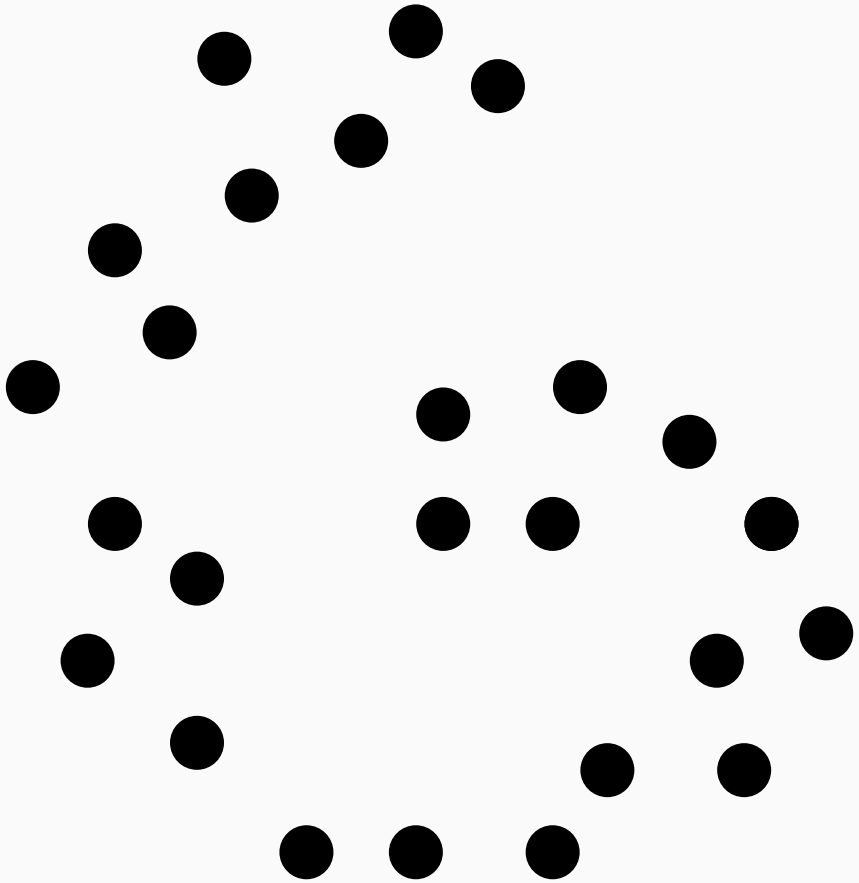


Figure 1: Example data

Isomap procedure

Step One: Use the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to build a graph G, drawing edges between "neighboring" nodes with each edge weighted by the Euclidean distance

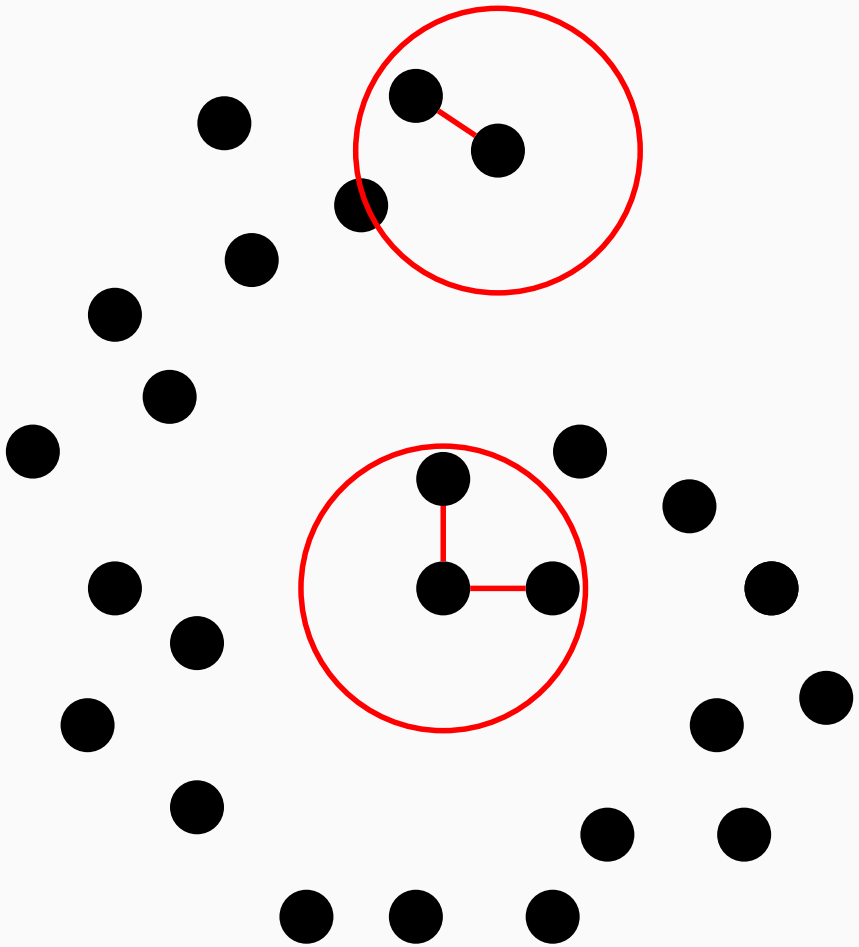
$$d_X(i,j) = ||\mathbf{x}_i - \mathbf{x}_j||_2$$

where "neighboring" nodes can be defined using one of the following methods.

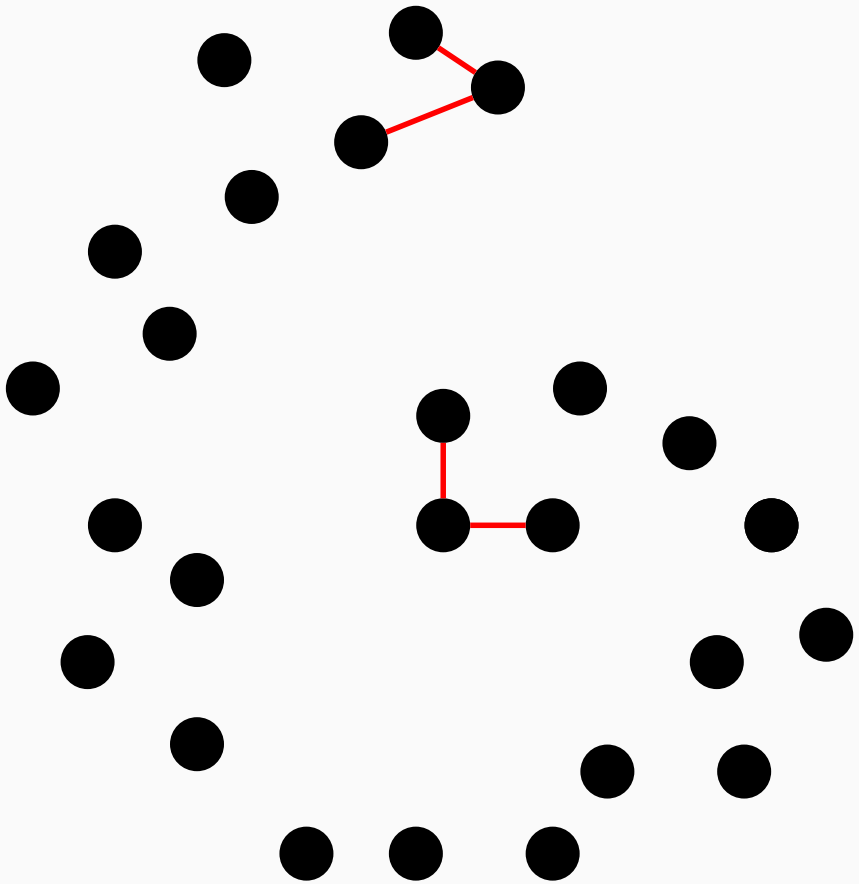
Two ways for constructing neighborhood graphs

1. **Using ϵ-balls**: For a given node $\mathbf{x}_i$, another node $\mathbf{x}_j$ is a neighbor if and only if $||\mathbf{x}_i - \mathbf{x}_j||_2 < \epsilon$.
2. **Using k nearest neighbors (k-NN)**: For a given node $\mathbf{x}_i$, the k closest nodes (in Euclidean distance) are neighbors.

Using ϵ-balls



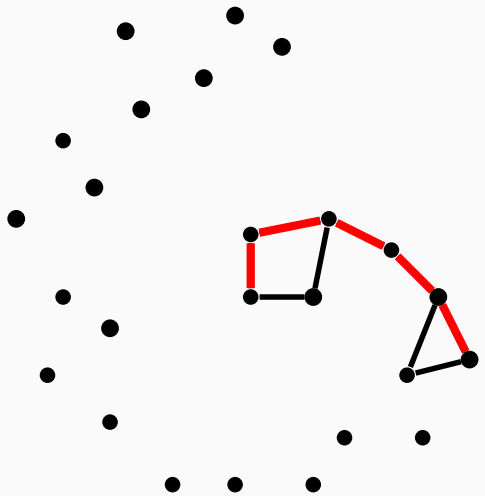
Using 2 nearest neighbors



Isomap procedure

Step Two: Given the graph G, we can compute the shortest path distance $d_G(i, j)$ between each pair of nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ using Dijkstra's algorithm.

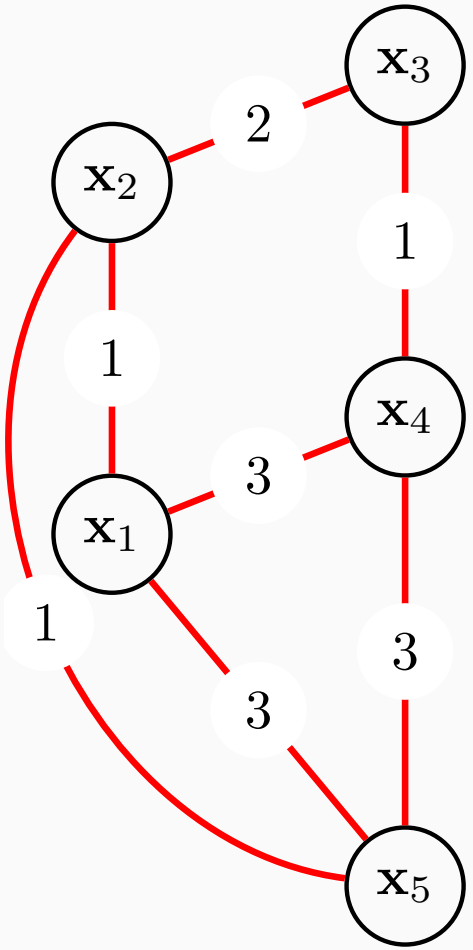
Example:



Dijkstra's algorithm

1. Select a starting node. Assign to every node a distance from the starting node, using 0 for the starting node and infinity for the others. Create a list of unvisited nodes, containing
2. Examine the neighbors of the starting node and update the distances. Mark the starting node as being visited.
3. Move on to the closest unvisited neighbor. Consider all of this node's neighbors and update the distances through the current node. Mark this as visited.
4. Move onto the unvisited node with the smallest known distance and repeat step 3.
5. Once all nodes have been visited, the shortest path can be retrieved by working backwards form the target node.

Dijkstra's algorithm



Isomap procedure

Step Three: Apply MDS (multidimensional scaling) with the matrix of shortest path distances $D = (d_G(i, j))$ to compute an embedding $\mathbf{y}_1, \dots, \mathbf{y}_n$.

Isomap procedure

Step One: Use the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to build a graph G, drawing edges between "neighboring" nodes with each edge weighted by the Euclidean distance

Step Two: Given the graph G, we can compute the shortest path distance $d_G(i, j)$ between each pair of nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ using Dijkstra's algorithm.

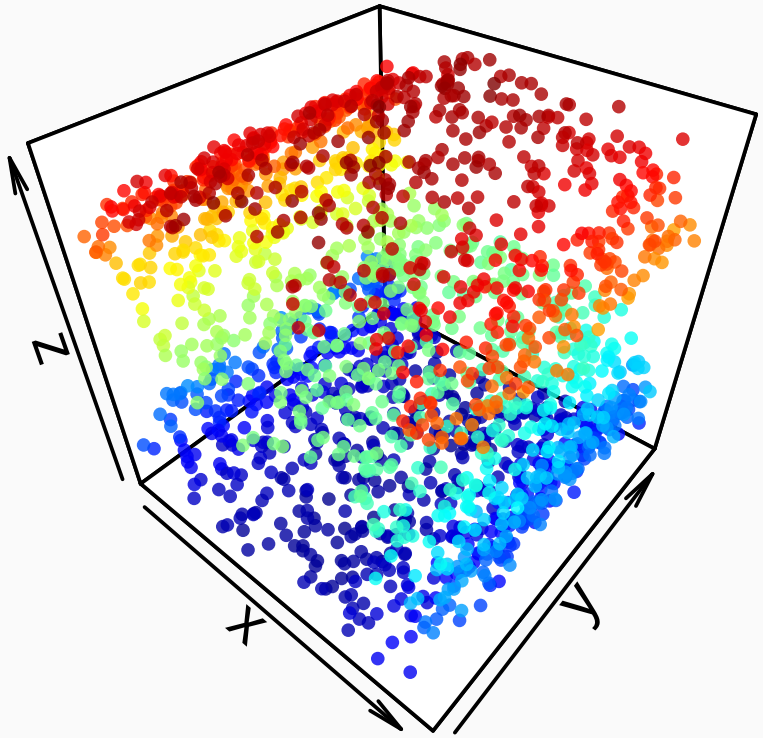
Step Three: Apply MDS (multidimensional scaling) with the matrix of shortest path distances $D = (d_G(i, j))$ to compute an embedding $\mathbf{y}_1, \ldots, \mathbf{y}_n$.

Key considerations

1. Building G may be computationally expensive/slow
2. Tuning parameters (such as ϵ or k) may significantly impact results
3. Problems can arise if the data has “holes” or “gaps.”

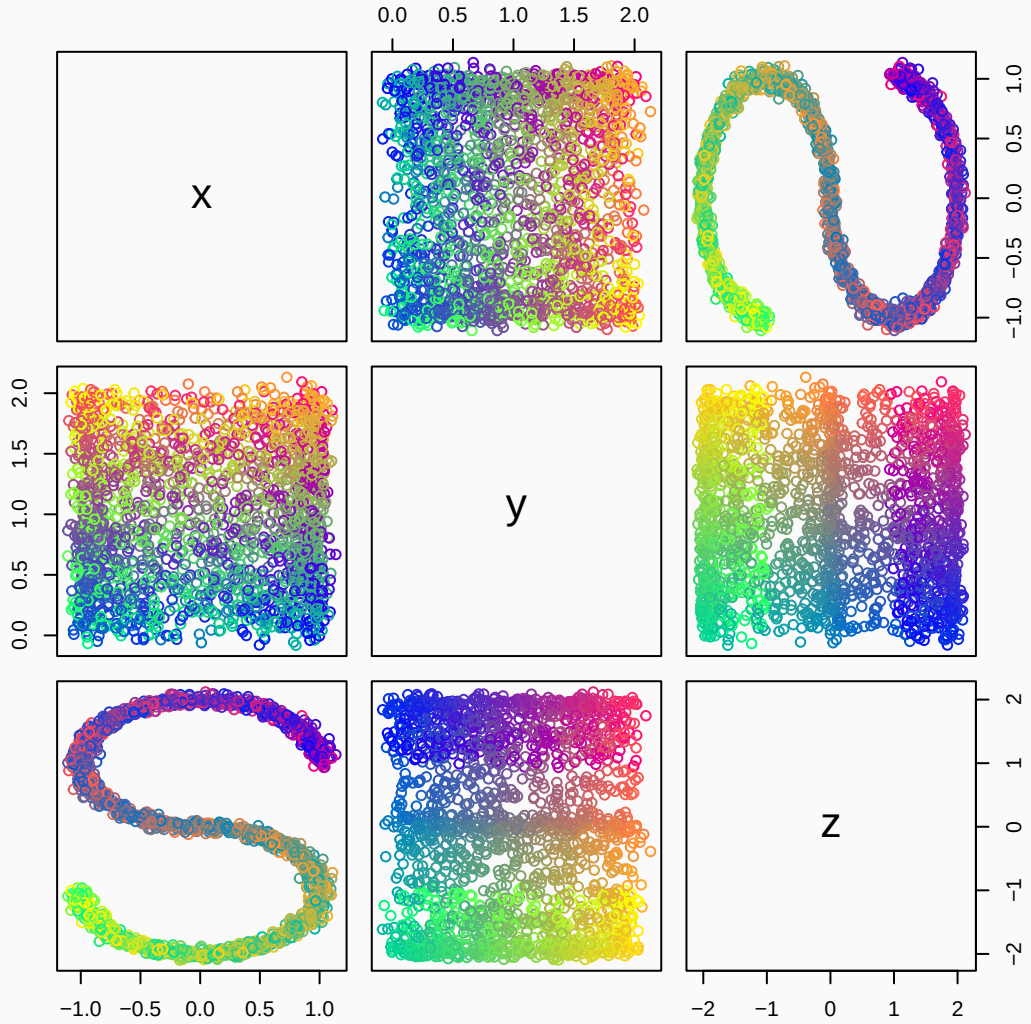
Example: 3D S curve

```
library(dimRed)
library(plot3D)
dat <- loadDataSet("3D S Curve", n = 2000)
scatter3D(x = dat@data[,1], y = dat@data[,2],
          z = dat@data[,3], pch = 16,
          alpha = .8, cex = .5, colkey = F, fill = T)
```



Example: 3D S curve

```
plot(dat)
```



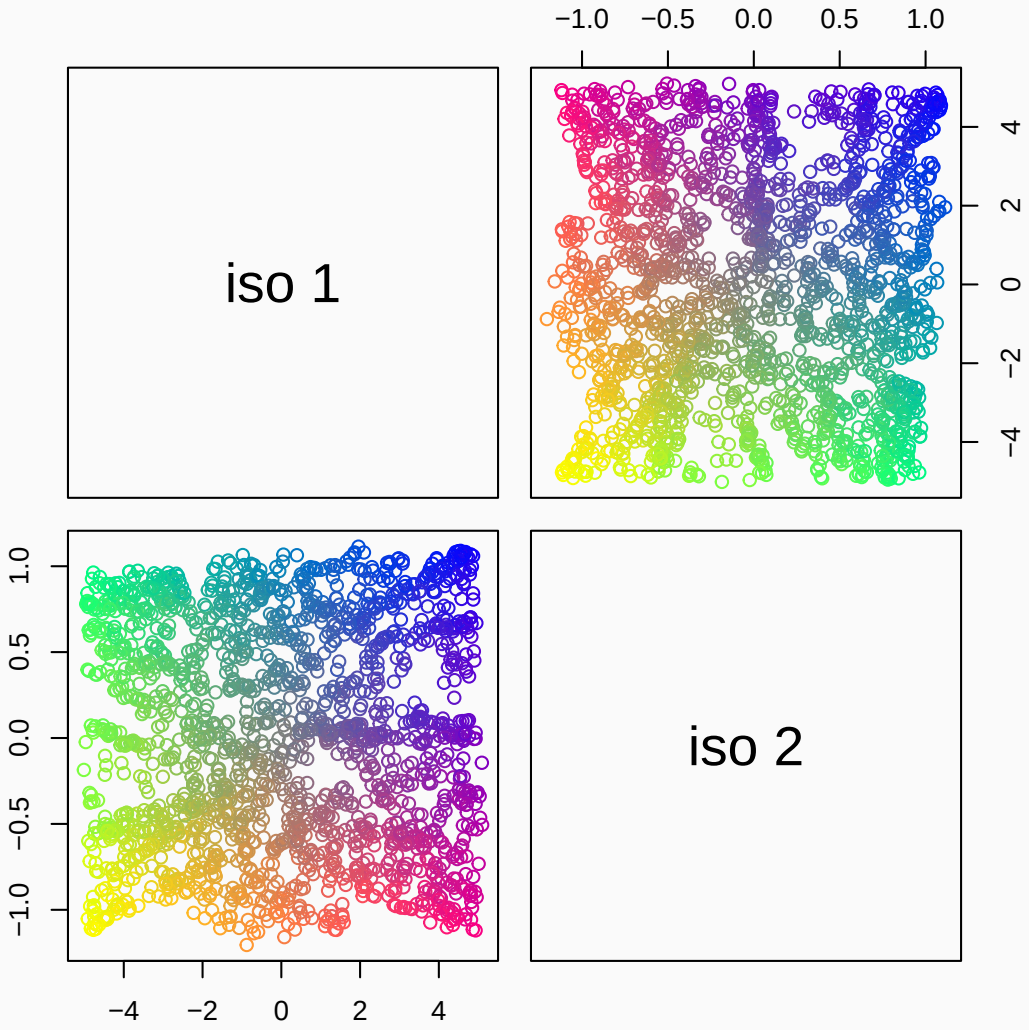
Example: 3D S curve

The `embed()` function can be used with the option `"Isomap"` to perform the Isomap procedure:

```
emb <- embed(dat, "Isomap", knn = 10, ndim = 2)
```

Example: 3D S curve

```
plot(emb)
```

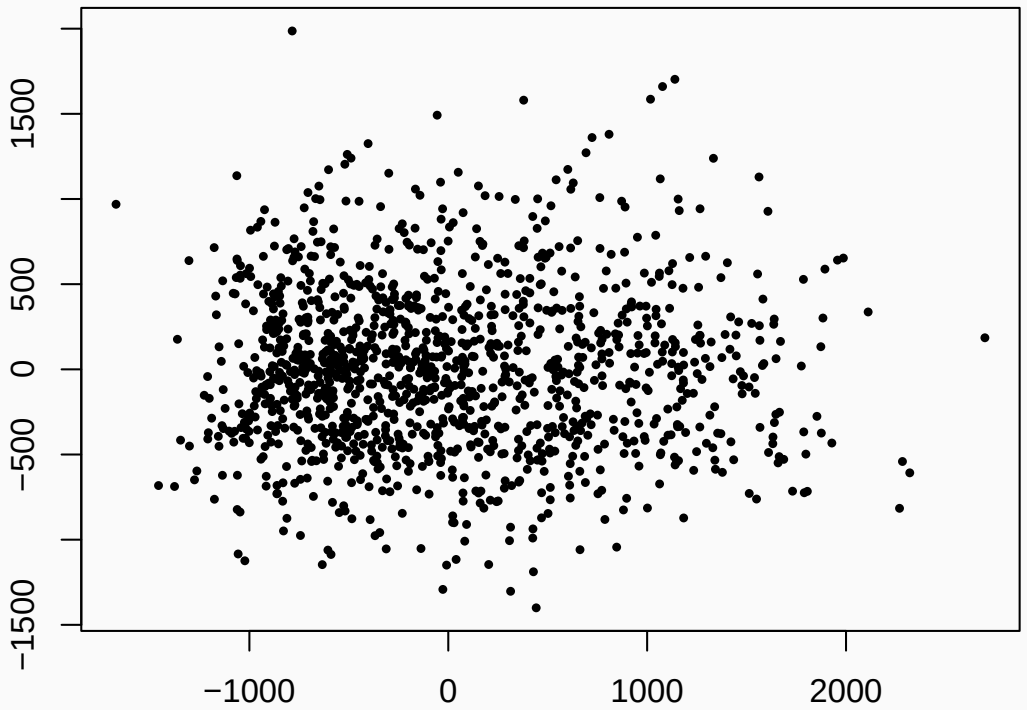


Example: USPS digits data

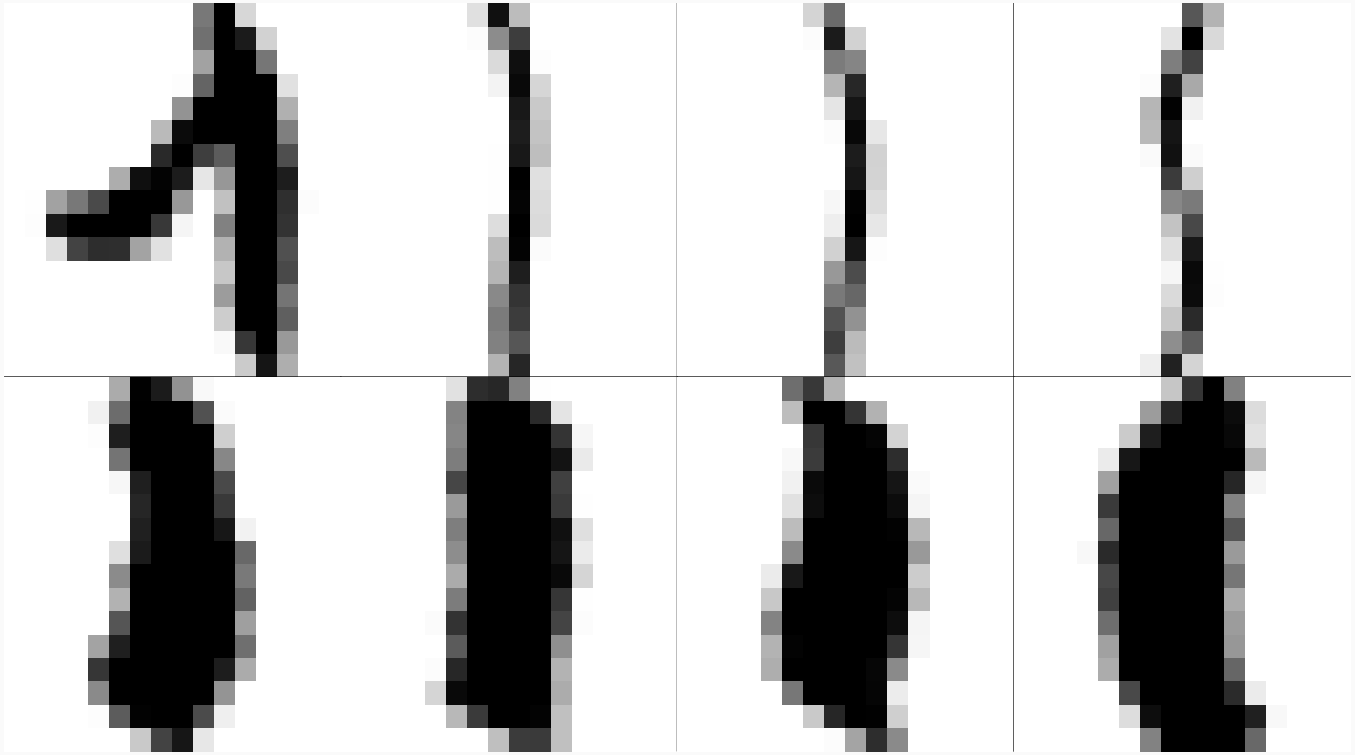
```
library(IMIFA)

data("USPSdigits")
usps <- list()
usps$data <- as.matrix(rbind(USPSdigits$train[, -1],
                             USPSdigits$test[, -1]))
usps$data <- (usps$data - -1) * 255 / 2
usps$label <- c(USPSdigits$train[, 1], USPSdigits$test[, 1])
usps$test <- c(rep(0, nrow(USPSdigits$train)),
               rep(1, nrow(USPSdigits$test)))

# split into test and training
train_ind <- usps$test == 0
usps_train <- usps$data[train_ind,]
usps_train_label <- usps$label[train_ind]
usps_test <- usps$data[!train_ind,]
usps_test_label <- usps$label[!train_ind]
usps_i <- usps$data[usps$label == 1, ]
```

Example: USPS digits data

```
emb <- embed(usps_i, "Isomap", knn = 10, ndim = 2)
plot(emb@data@data[, 1], emb@data@data[, 2], pch = 16,
     cex = .6, xlab = "", ylab = "")
```



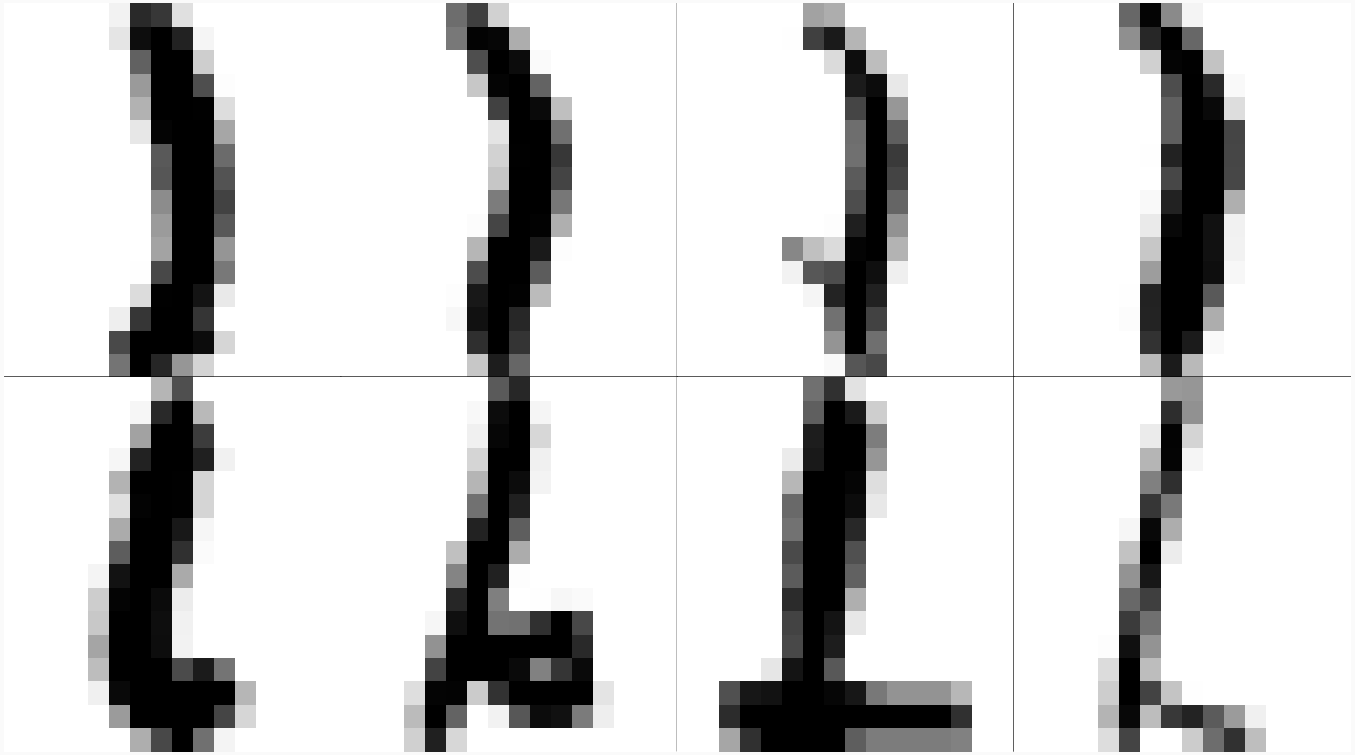
Example: USPS digits data



Example: USPS digits data

```
top_d1 <-head(sort(emb@data@data[, 1], index.return = T)$ix, 4)
bottom_d1 <-tail(sort(emb@data@data[, 1], index.return = T)$ix, 4)
par(mfrow = c(2, 4))
par(mar = c(0,0,0,0))
for (j in c(top_d1, bottom_d1)) {
  image(matrix(usps_i[j, ],nrow=16)[,16:1],
        col = gray((255:0)/255), yaxt = "n", xaxt = "n")
}
```

Example: USPS digits data



Example: USPS digits data

```
top_d2 <- head(sort(emb@data@data[, 2], index.return = T)$ix, 4)
bottom_d2 <- tail(sort(emb@data@data[, 2], index.return = T)$ix, 4)
par(mfrow = c(2, 4))
par(mar = c(0,0,0,0))
for (j in c(top_d2, bottom_d2)) {
  image(matrix(usps_i[j, ],nrow=16)[,16:1],
        col = gray((255:0)/255), yaxt = "n", xaxt = "n")
}
```

One-shot algorithms

Isomap is an example of what Meila and Zhang call a one-shot algorithm, which identifies an embedding by decomposing some matrix associated with the neighborhood graph.

In this context, **one-shot** refers to the fact that estimation is not **iterative**–we can simply compute a single matrix decomposition or solve some optimization problem.

Diffusion Maps

Another one-shot embedding algorithm is Diffusion Mapping (Coifman and Lafon 2006).

Like Isomap, diffusion mapping starts by identifying the neighborhood graph G.

A detour: heat equation

The heat equation is a certain partial differential equation originally developed to model the diffusion of heat.

We say u is a solution of the heat equation if

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_p^2} = \Delta u$$

where Δ denotes the **Laplacian** operator.

Intuitively, the heat equation says the rate at which heat u changes is proportional to how much hotter (or cooler) the local environment is.

Relating diffusion and distance

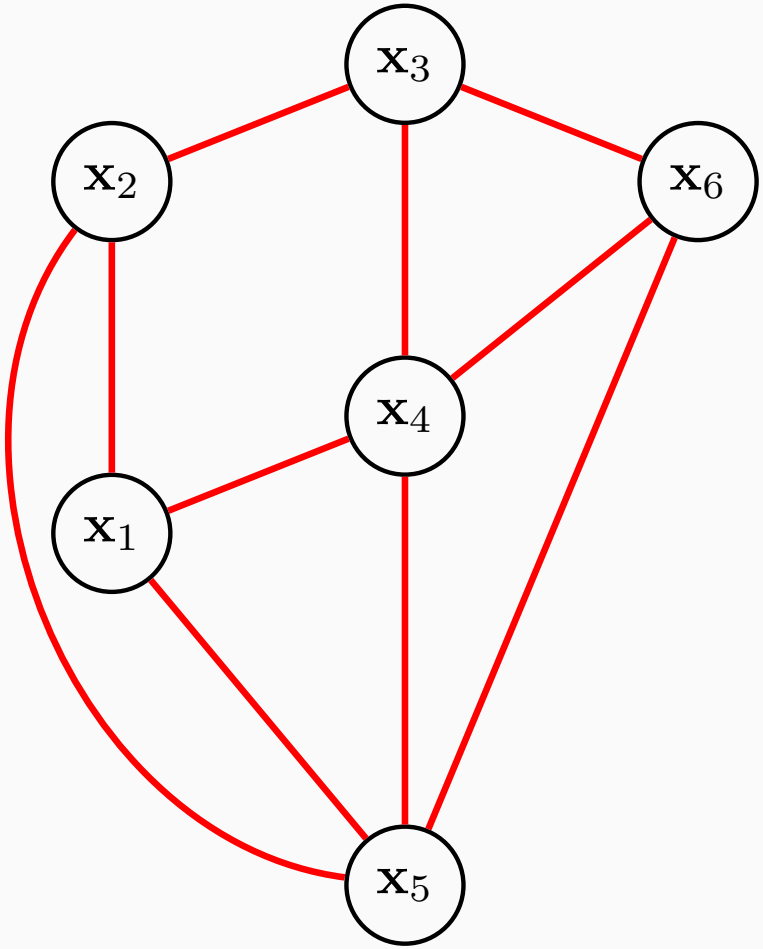
Manifold learning attempts to embed high dimensional data in a way that preserves geometric structure. For example, Isomap attempts to preserve geodesic distances between points.

The heat equation and other diffusion process imply that heat/other quantities diffuse more quickly between points that are close by or similar.

Given data $\mathbf{x}_1, \dots, \mathbf{x}_n$, we will use diffusion processes to calculate the pairwise "diffusion distances" between each pair of points. These diffusion distances can be used to develop a low-dimensional embedding $\mathbf{y}_1, \dots, \mathbf{y}_n$.

Random walk on the graph G

As with Isomap, diffusion mapping begins by identifying a neighborhood graph either by ϵ-balls or k-NN:



Random walk on the graph G

We can construct a random walk process based on this graph, where for each pair of **neighbors** $\mathbf{x}_i, \mathbf{x}_j$, we can define the connectivity between $\mathbf{x}_i$ and $\mathbf{x}_j$ to be the probability of jumping from $\mathbf{x}_i$ to $\mathbf{x}_j$.

Usually, we model the connectivity using a kernel function

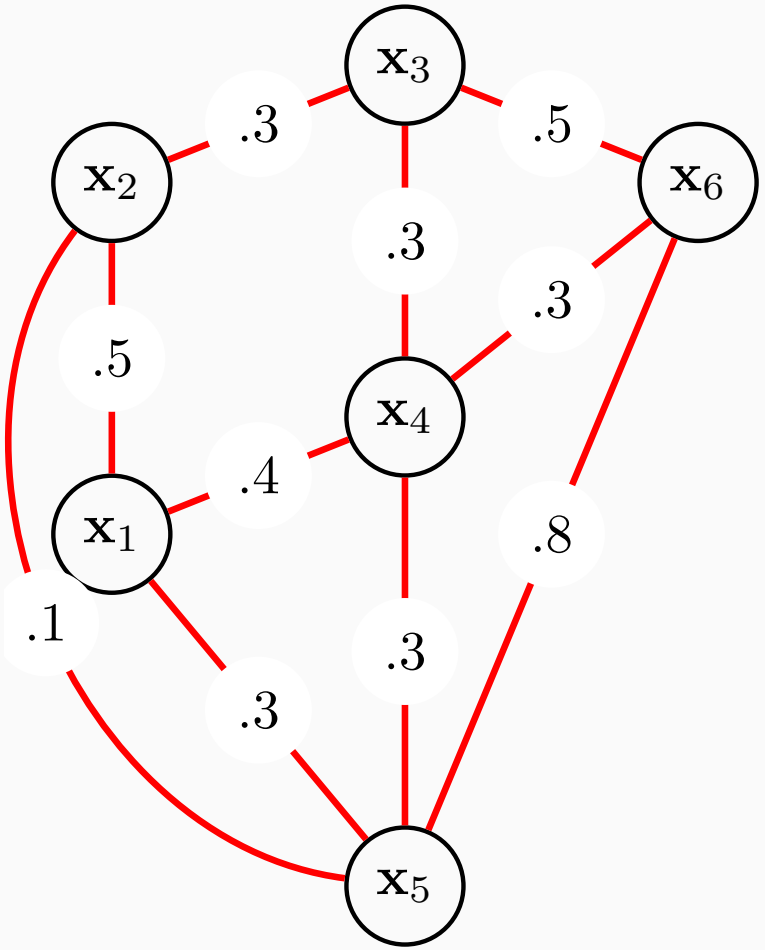
$$k(\mathbf{x}_i, \mathbf{x}_i) \propto \text{connectivity}(\mathbf{x}_i, \mathbf{x}_i)$$

Unless otherwise specified, we will assume that k is the Gaussian kernel

$$k(\mathbf{x}_i, \mathbf{x}_i) = \exp\left(\frac{||\mathbf{x}_i - \mathbf{x}_j||_2}{h^2}\right)$$

with some kernel width h that controls the scaling of the connectivity.

Random walk on the graph G



Connectivity/Similarity matrix

We can then compute $K = (k_{ij}) = (k(\mathbf{x}_i, \mathbf{x}_j))$

Note that for non-neighboring pairs, we define $k_{ij} = 0$. Similarly, the diagonal entries $k_{ii} = 0$.

The matrix K thus encodes information about the pairwise similarities/connectivities of the graph G.

Graph definitions

We define a **weighted graph** to consist of a set of vertices $V = \{1, \dots, n\}$, edges E, and a similarity matrix K, $G = (V, E, K)$, where $k_{ij} \geq 0$ denotes the weight for the edge between i and j ($k_{ij} = 0$ if and only if there is no edge between i and j.)

The **degree** of a vertex i is defined as $d_i = \sum_{j=1}^{n} k_{ij}$ and quantifies the level o connectivity between i and the rest of the graph.

Graph definitions

A **subgraph** of G is another graph formed by taking a subset of vertices $A \subset V$ and keeping any edges connecting pairs of vertices in A.

A **path** is a sequence of vertices and edges in which no vertex or edge is repeated.

A subgraph is **connected** if any two vertices in A can be joined by a path only using edges/vertices in A.

A subgraph is a **connected component** if it is connected and there are no edges between A and its complement $V - A$.

A graph is **connected** if it only has one connected complement.

Graph Laplacian

Let $d_i = \sum_{j=1}^{n} k_{ij}$ be the **degree** of node i (equal to the ith row sum of K).

Let D be the diagonal matrix with entries $d_1, \ldots, d_n$.

The **graph Laplacian** matrix is defined as

$$L = D - K$$

Graph Laplacian

Suppose we have a real-valued function f on the graph (so f maps every node $\mathbf{x}_i$ to a real number).

We can think of f as a vector $\mathbf{f} \in \mathbb{R}^n$ with $L\mathbf{f}$ is a vector with ith element (**exercise**):

$$(L\mathbf{f})_i = \sum_j k_{ij}(f_i - f_j)$$

Moreover (**exercise**):

$$\mathbf{f}^\top L\mathbf{f} = \frac{1}{2}\sum_i \sum_j k_{ij}(f_i - f_j)^2$$

What does it mean when $\mathbf{f}^\top L\mathbf{f}$ is large? When $\mathbf{f}^\top L\mathbf{f}$ is small?

Properties of the graph Laplacian

Observe that $L \in \mathbb{R}^{n \times n}$ and:

- L is symmetric
- All the rows and columns of L sum to 0, meaning that $\mathbf{1}$ is an eigenvector of L with eigenvalue 0.
- For any vector $\mathbf{f} \in \mathbb{R}^n$,

$$\mathbf{f}^\top L \mathbf{f} = \frac{1}{2} \sum_i \sum_j k_{ij} (f_i - f_j)^2$$

  meaning that L is positive semidefinite. Thus, its eigenvalues are nonnegative.
- The algebraic multiplicity of the eigenvalue 0 is the number of connected components of the graph.

Properties of the graph Laplacian

We can view the graph Laplacian matrix as a matrix version of the negative discrete Laplace operator on a graph. That is, observe that for any $\mathbf{f} \in \mathbb{R}^n$:

$$L\mathbf{f} = \frac{1}{2} \sum_i \sum_j k_{ij}(f_i - f_j)^2 \approx -\Delta f$$

where Δf represents the Laplacian (on a continuous domain without boundaries) applied to a function f.

Diffusion mapping in one dimension

Suppose we wish to construct a one-dimensional embedding of our data $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

Equivalently, we wish to choose a vector $\mathbf{f} = f_1, \ldots, f_n$ where $f_i = f(\mathbf{x}_i)$ to represent our data.

Diffusion mapping: Choose $\mathbf{f}$ to satisfy

$$\mathbf{f} = \arg\min_{\mathbf{f} \in \mathbb{R}^n} \mathbf{f}^\top L \mathbf{f} = \arg\min_{\mathbf{f} \in \mathbb{R}^n} \frac{1}{2} \sum_i \sum_j k_{ij} (f_i - f_j)^2$$

Diffusion mapping in one dimension

Interpretation:

- Minimizing this quantity requires minimizing $f_i - f_j$ when k_{ij} are large.
- When k_{ij} is small, $f_i - f_j$ contributes less to $\mathbf{f}^\top L\mathbf{f}$, so differences are less penalized.

Diffusion mapping in one dimension

Note that the optimization problem

$$\min_{\mathbf{f}\in\mathbb{R}^n} \mathbf{f}^\top L\mathbf{f} = \min_{\mathbf{f}\in\mathbb{R}^n} \frac{1}{2}\sum_i \sum_j k_{ij}(f_i - f_j)^2$$

is not well-defined yet.

In particular, there are a number of trivial solutions we wish to avoid (exercise).

Diffusion mapping in one dimension

What happens if we replace f_i and f_j by $f_i + c$ and $f_j + c$ (translation)?

What happens if $f_i = f_j$ for all $i \neq j$?

To deal with this invariance and to eliminate the trivial solutions, we must add some constraints to $\mathbf{f}$ to obtain a unique solution.

Diffusion mapping in one dimension

In particular, we require:

- a sum-to-zero constraint $\mathbf{f}^\top \mathbf{1} = \sum_i f_i = 0$
- the norm of $\mathbf{f}$ to be 1: $||\mathbf{f}||_2 = 1$

Equivalently, we can rewrite our minimization problem as

$$\min_{\mathbf{f} \neq \mathbf{0}, \mathbf{f}^\top \mathbf{1} = 0} \frac{\frac{1}{2} \sum_i \sum_j k_{ij} (f_i - f_j)^2}{\sum_i f_i^2} = \min_{\mathbf{f} \neq \mathbf{0}, \mathbf{f}^\top \mathbf{1} = 0} \frac{\mathbf{f}^\top L \mathbf{f}}{\mathbf{f}^\top \mathbf{f}}$$

Anything look familiar?

Diffusion mapping in one dimension

The problem

$$\min_{\mathbf{f}\neq\mathbf{0},\mathbf{f}^\top\mathbf{1}=0} \frac{\mathbf{f}^\top L\mathbf{f}}{\mathbf{f}^\top\mathbf{f}}$$

is a (constrained) Rayleigh quotient problem.

Without the constraint $\mathbf{f}^\top\mathbf{1} = 0$ the minimizing $\mathbf{f}$ is the eigenvector of L corresponding to the smallest eigenvalue $\lambda_1 = 0$.

In particular the eigenvector is $\mathbf{v}_1 = \mathbf{1}$.

Diffusion mapping in one dimension

Using the constraint, we choose $\mathbf{f}$ to be orthogonal to $\mathbf{1}$.

The minimizer of the constrained Rayleigh quotient is given by the eigenvector corresponding to the second smallest eigenvalue λ_2:

$$\mathbf{f} = \mathbf{v}_2$$

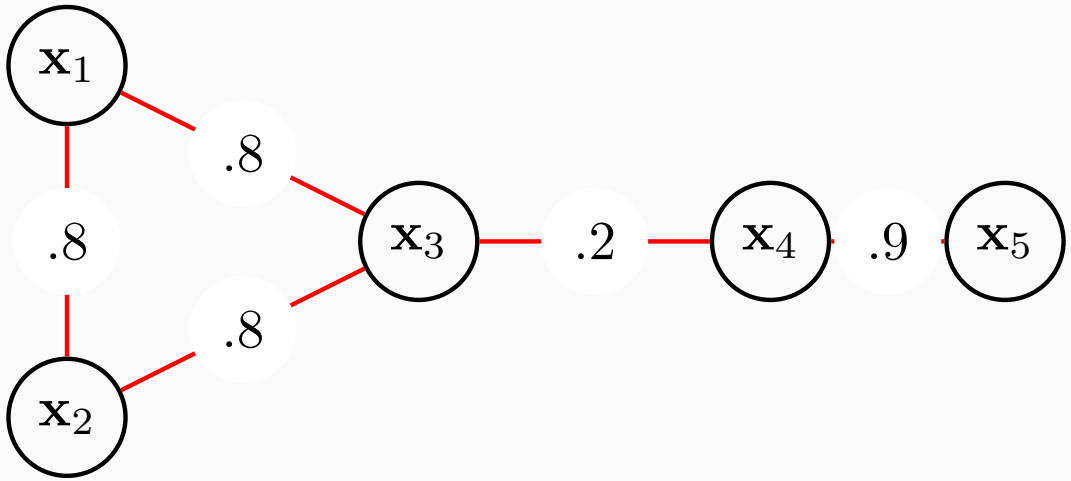
Diffusion mapping in one dimension

It turns out that if the graph G is **connected**, then the algebraic multiplicity of the eigenvalue $\mathbf{0}$ is one. In this case, $\lambda_2 > 0$ and

$$0 < \mathbf{f}^\top L\mathbf{f} = \frac{1}{2}\sum_i \sum_j k_{ij}(f_i - f_j)^2$$

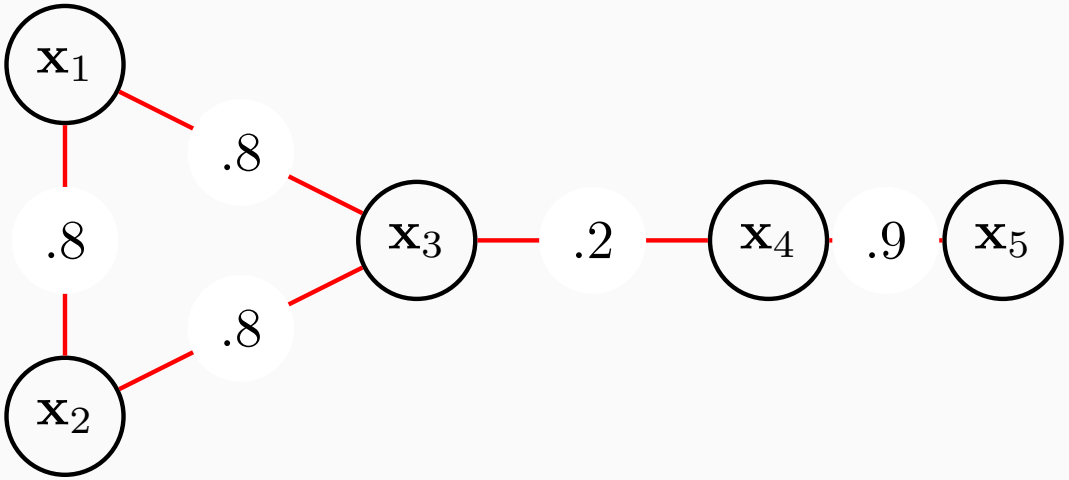
indicating that $\mathbf{f} = \mathbf{v}_2$ is a non-trivial embedding of $\mathbf{x}_1, \dots, \mathbf{x}_n$.

Example: 1D Diffusion Mapping



$$L = D - W = \begin{bmatrix} 1.6 & 0 & 0 & 0 & 0 \\ 0 & 1.6 & 0 & 0 & 0 \\ 0 & 0 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1.1 & 0 \\ 0 & 0 & 0 & 0 & 0.9 \end{bmatrix} - \begin{bmatrix} 0 & .8 & .8 & 0 & 0 \\ .8 & 0 & .8 & 0 & 0 \\ .8 & .8 & 0 & .2 & 0 \\ 0 & 0 & .2 & 0 & .9 \\ 0 & 0 & 0 & .9 & 0 \end{bmatrix}$$

Example: 1D Diffusion Mapping



$$L = \begin{bmatrix} 1.6 & -0.8 & -0.8 & 0 & 0 \\ -0.8 & 1.6 & -0.8 & 0 & 0 \\ -0.8 & -0.8 & 1.8 & -0.2 & 0 \\ 0 & 0 & -0.2 & 1.1 & -0.9 \\ 0 & 0 & 0 & -0.9 & 0.9 \end{bmatrix}$$

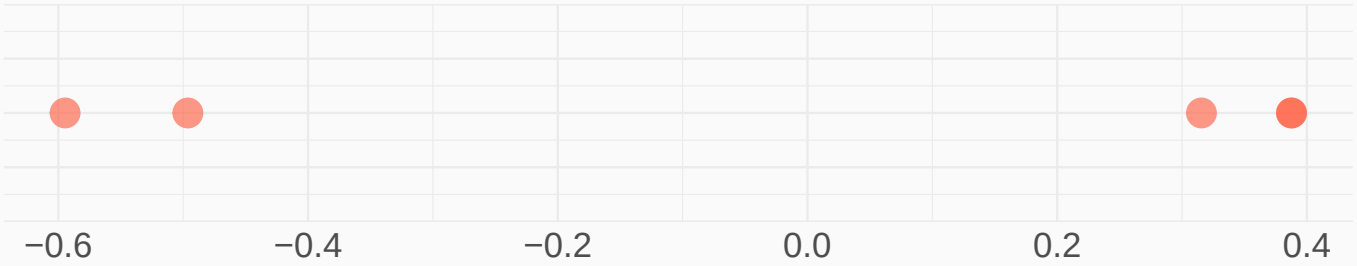
Example: 1D Diffusion Mapping

For this graph Laplacian, the eigenvalues are

$$0 < 0.15, 1.89 < 2.40 < 2.57$$

The corresponding eigenvector is

$$\mathbf{v}_2 = (0.39, 0.39, 0.32, -0.50, -0.59)$$



Example: Code for 1D Diffusion Mapping

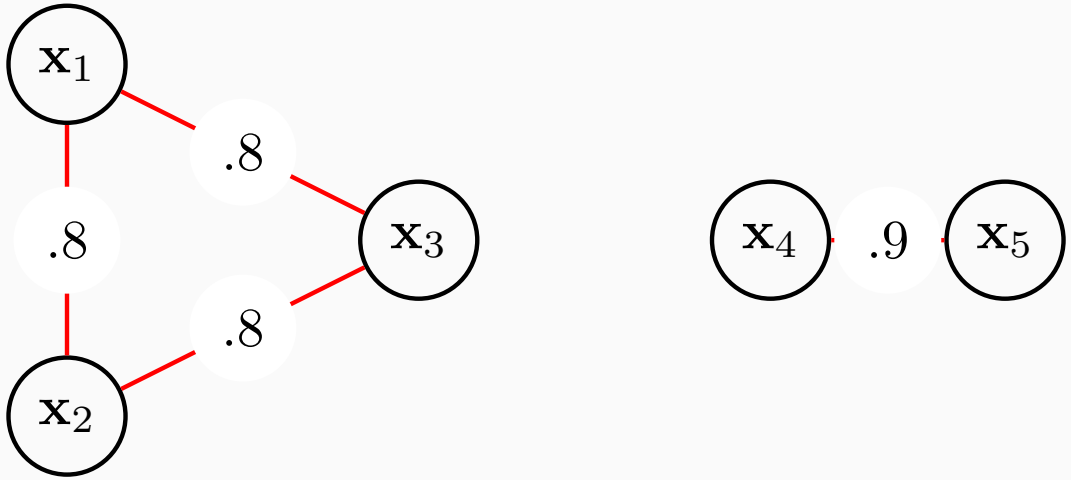
```
library(igraph)
adj_mat <- matrix(
  c(0, .8, .8, 0, 0, .8, 0, .8, 0, 0,
    .8, .8, 0, .2, 0, 0, 0, .2, 0, .9,
    0, 0, 0, .9, 0),
  nrow = 5
)
example_graph <-
  graph_from_adjacency_matrix(adj_mat,
                              mode = "undirected",
                              weighted = T)
L <- graph.laplacian(example_graph)
library(ggplot2)
ggplot() +
  geom_point(aes(x = eigen(L)$vectors[, 4], y = 0),
             alpha = .65, cex = 6, color = 'tomato') +
  xlab("") + ylab("") +
  theme_minimal() +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 16))
```

Diffusion mapping: q-dimensional embedding

Taking the q eigenvectors corresponding to the q smallest non-zero eigenvalues yields a higher dimensional embedding.

In particular, represent $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as $\mathbf{y}_1, \ldots, \mathbf{y}_n$ where $\mathbf{y}_i = (v_{1i}, \ldots, v_{qi})^\top$.

Example: Disconnected graph



$$L = \begin{bmatrix} 1.6 & -0.8 & -0.8 & 0 & 0 \\ -0.8 & 1.6 & -0.8 & 0 & 0 \\ -0.8 & -0.8 & 1.6 & 0 & 0 \\ 0 & 0 & 0 & .9 & -0.9 \\ 0 & 0 & 0 & -0.9 & 0.9 \end{bmatrix}$$

Now L has a repeated eigenvalue 0. In this case, using more than one dimension is necessary.

Graph Laplacian

In practice, Coifman and Lafon (2006) use a scaled version of L.

1. First, normalize columns by computing $k_{\cdot j} = \sum_i k_{ij}$ and calculate $\hat{k}_{ij} = k_{ij}/k_{\cdot j}$.
2. Normalize rows by computing $\hat{k}_{i\cdot} = \sum_j \hat{k}_{ij}$ and calculate $p_{ij} = \hat{k}_{ij}/\hat{k}_{i\cdot}$.
3. Define $L = (I - P)/h^2$, where $P = (p_{ij})$ and h is the kernel width.

This normalization removes bias related to the density of the neighbor graph G.

Laplacian Eigenmaps

Before Coifman and Lafon (2006), Belkin and Niyogi (2003) proposed the use of a normalized version of L:

$$L^{norm} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} K D^{-1/2}$$

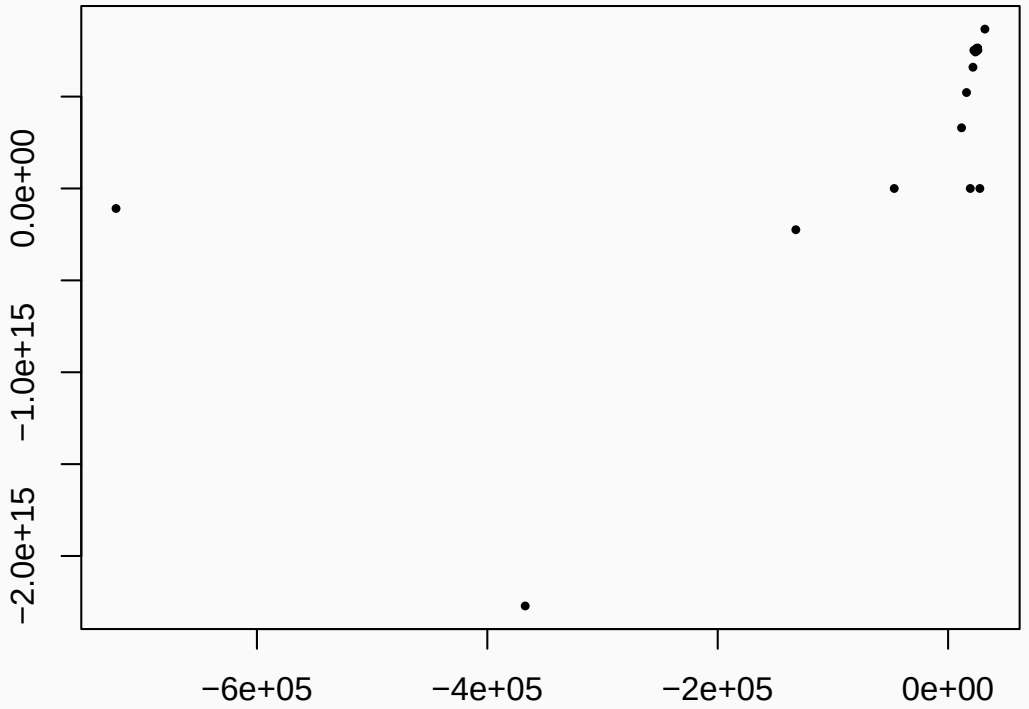
Observe that if the degrees of the nodes are constant $L = L^{norm}$. However, Coifman and Lafon (2006) claim that using L^{norm} yields eigenvectors that are influenced by the graph density.

Example: USPS digits data

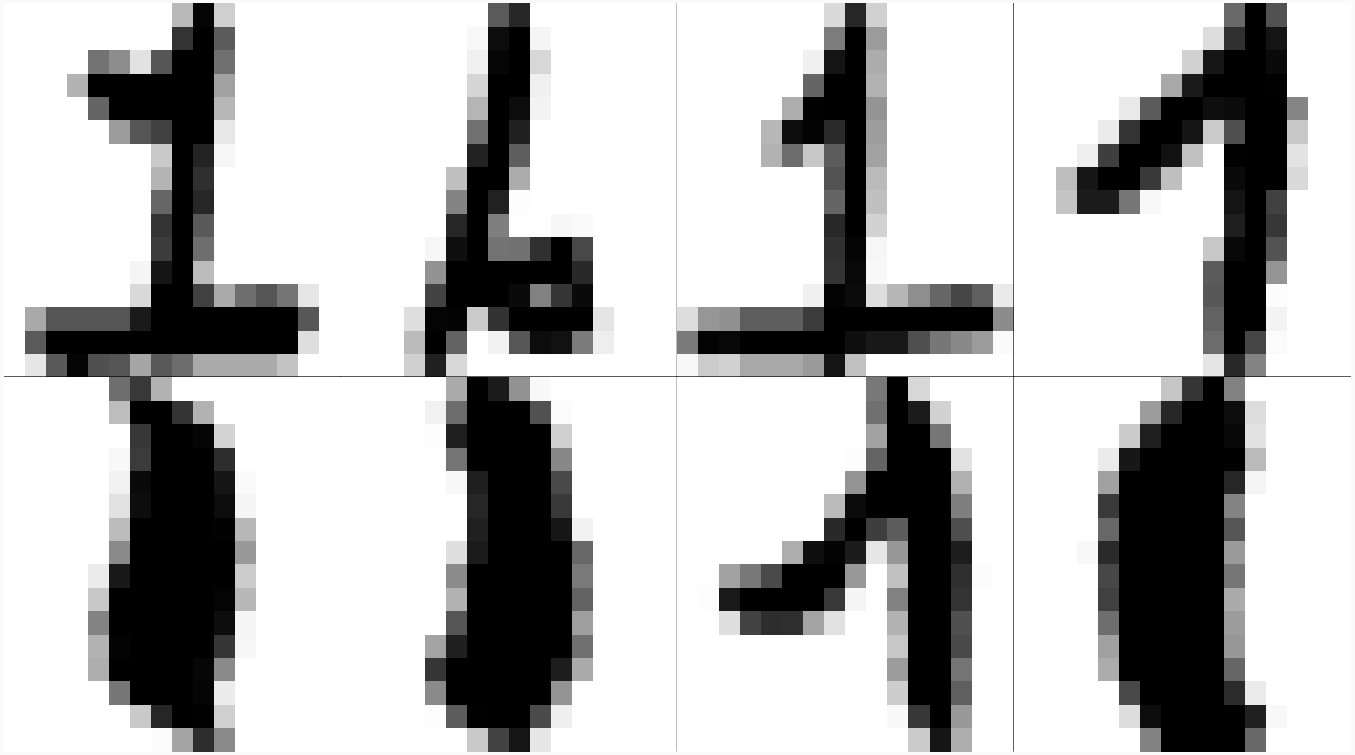
```
embDM <- embed(usps_i, "DiffusionMaps")
```

```
Performing eigendecomposition
Computing Diffusion Coordinates
Elapsed time: 9.498 seconds
```

```
plot(embDM@data@data[, 1], embDM@data@data[, 2], pch = 16,
     cex = .6, xlab = "", ylab = "")
```

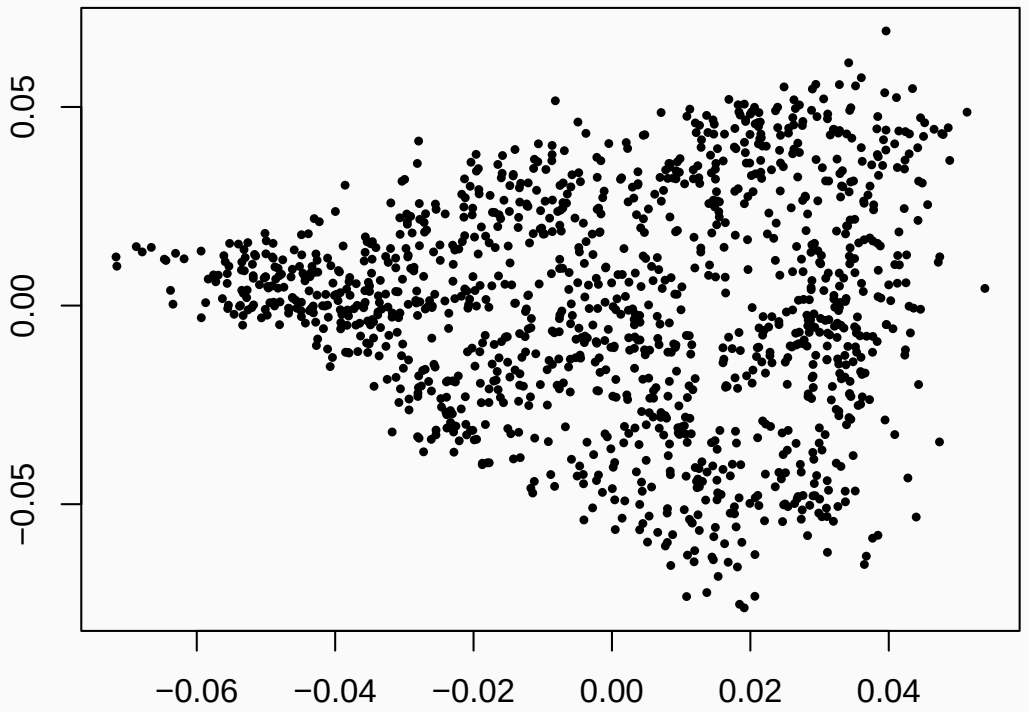


Example: USPS digits data

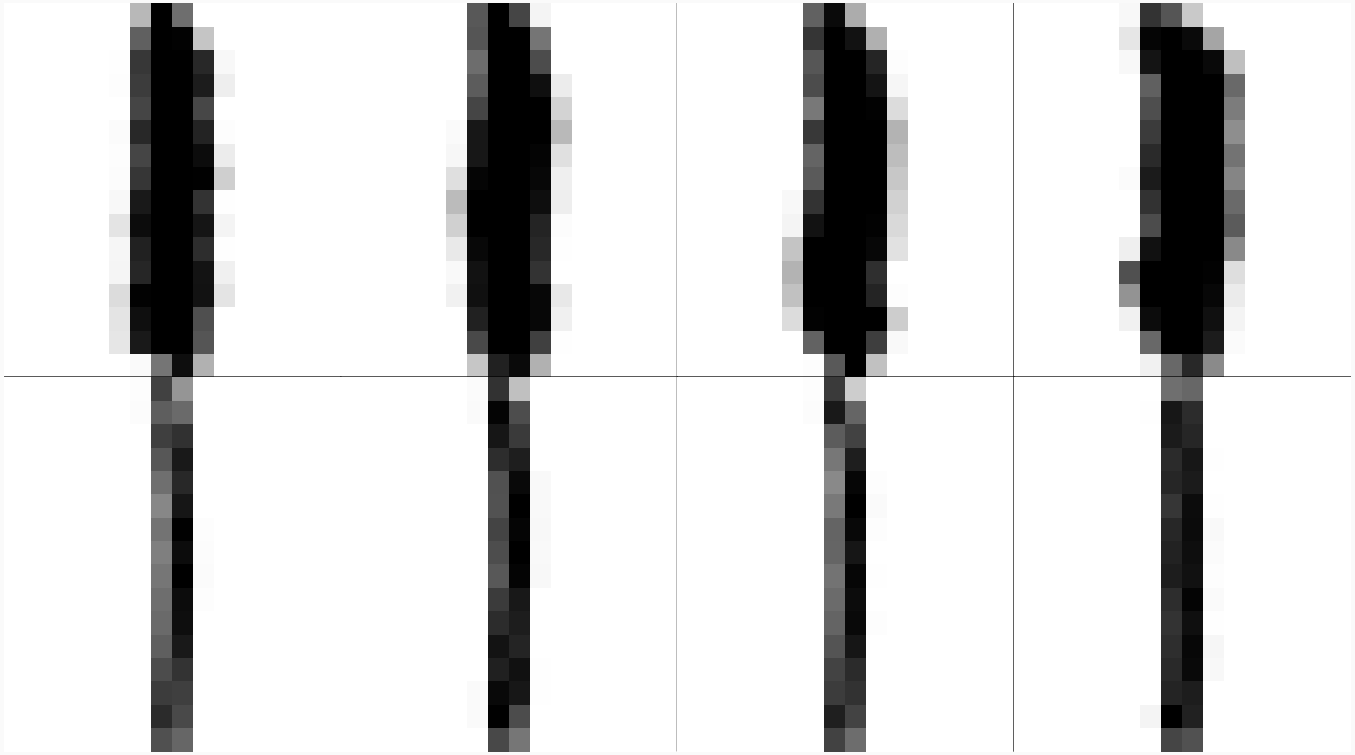


Example: USPS digits data

```
embLE <- embed(usps_i, "LaplacianEigenmaps", knn = 10, ndim = 2)
plot(embLE@data@data[, 1], embLE@data@data[, 2], pch = 16,
     cex = .6, xlab = "", ylab = "")
```



Example: USPS digits data

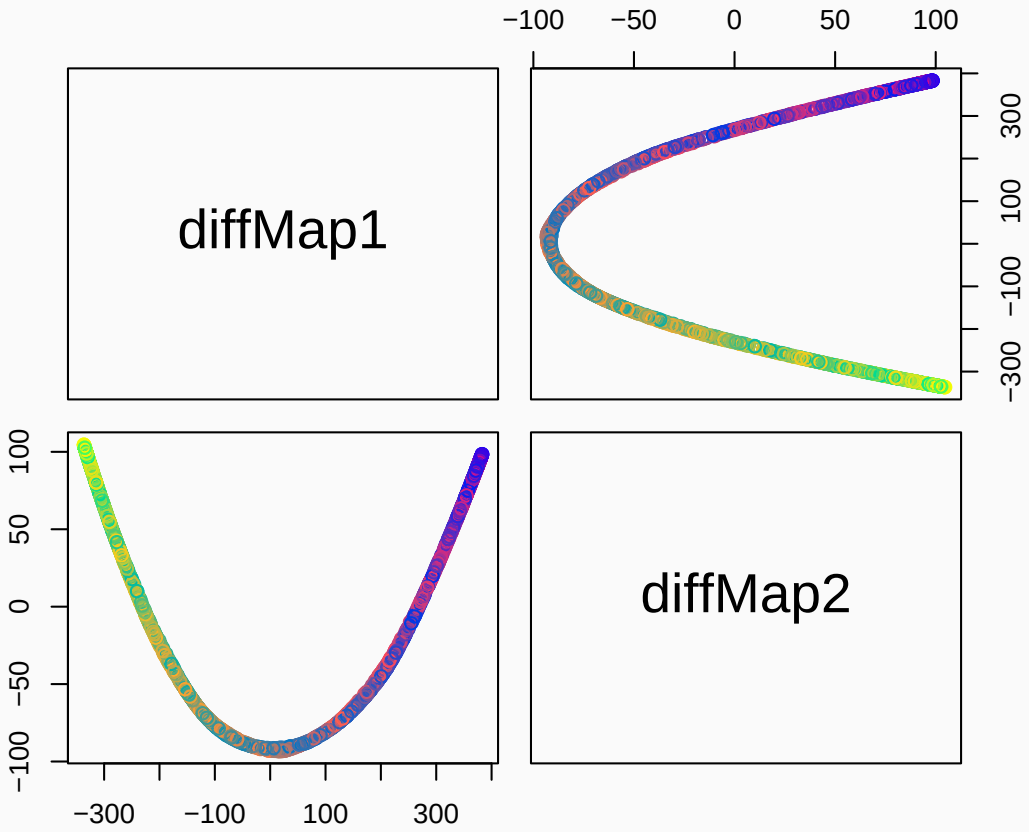


Example: 3D Swiss Roll

```
embDM <- embed(dat, "DiffusionMaps", knn = 10, ndim = 2)
```

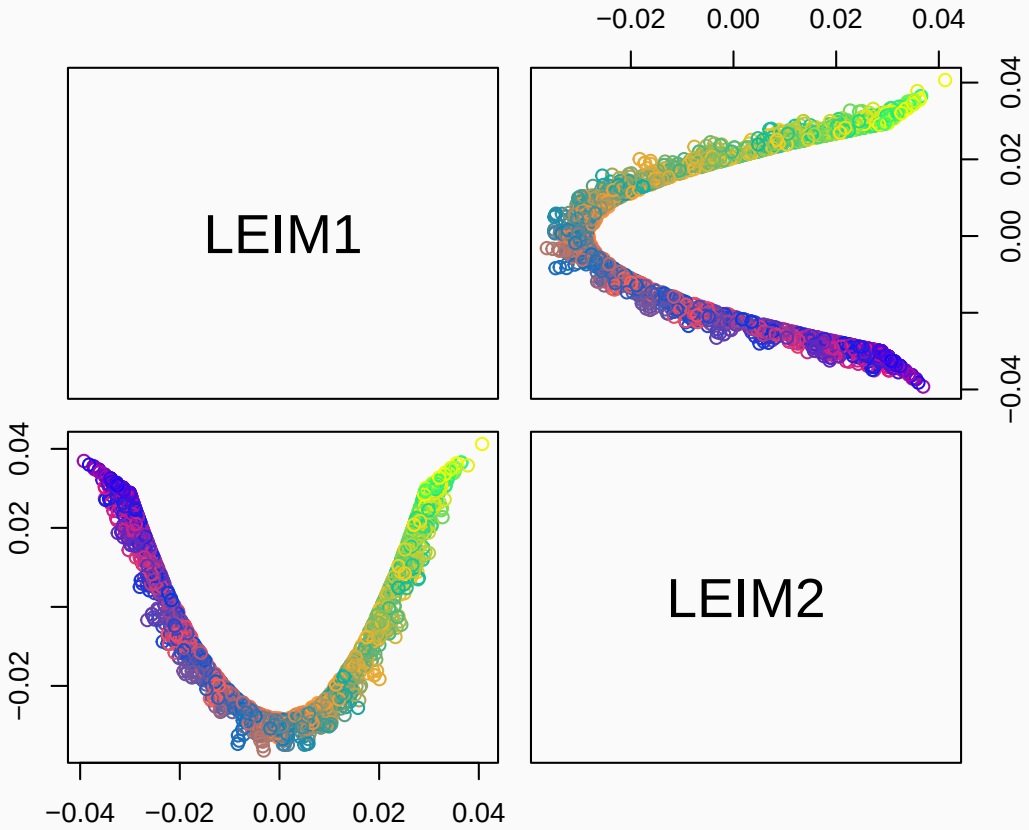
```
Performing eigendecomposition
Computing Diffusion Coordinates
Elapsed time: 0.417 seconds
```

```
plot(embDM)
```



Example: 3D Swiss Roll

```
embLE <- embed(dat, "LaplacianEigenmaps", knn = 10, ndim = 2)
plot(embLE)
```



Summary

In PCA, the data are projected onto principal directions that represent directions of maximum variance.

Isomap constructs an embedding using MDS with estimated geodesic distances..

For diffusion mapping the embedding represents the smoothest eigenvectors of L, the graph Laplacian.

Pitfalls

Laplacian eigenmaps tend to shrink areas where data is dense and stretch areas where data is sparse. As a result, clusters can be exaggerated.

Diffusion mapping should address this using renormalization.

References

Belkin, Mikhail, and Partha Niyogi. 2003. “Laplacian Eigenmaps for Dimensionality Reduction and Data Representation.” *Neural Computation* 15 (6): 1373–96. https://doi.org/10.1162/089976603321780317.

Coifman, Ronald R., and Stéphane Lafon. 2006. “Diffusion Maps.” *Applied and Computational Harmonic Analysis*, Special Issue: Diffusion Maps and Wavelets, 21 (1): 5–30. https://doi.org/10.1016/j.acha.2006.04.006.

Kraemer, Guido, Markus Reichstein, and Miguel D. Mahecha. 2018. “dimRed and coRanking - Unifying Dimensionality Reduction in R.” *The R Journal* 10 (1): 342–58. https://journal.r-project.org/archive/2018/RJ-2018-039/index.html.

Meilă, Marina, and Hanyu Zhang. 2024. “Manifold Learning: What, How, and Why.” *Annual Review of Statistics and Its Application* 11 (1): annurev-statistics-040522-115238. https://doi.org/10.1146/annurev-statistics-040522-115238.

Tenenbaum, Joshua B., Vin de Silva, and John C. Langford. 2000. “A Global Geometric Framework for Nonlinear Dimensionality Reduction.” *Science* 290 (5500): 2319–23. https://doi.org/10.1126/science.290.5500.2319.