Lecture 42 - Manifold Learning: ISOMAP & LLE

MultiDimensional Scaling (MDS)

Another linear technique with a similar aim is **multidimensional scaling**, or **MDS**. It finds a low-dimensional projection of the data such as to preserve, as closely as possible, the pairwise distances between data points, and involves finding the eigenvectors of the distance matrix. In the case where the distances are Euclidean, it gives equivalent results to PCA. Therefore, MDS is a generalization of PCA.

Given an assumed Euclidean proximity matrix, D, the **goal** of MDS is to find a set of points, Y, that have the same proximity matrix in an M-dimensional space, where $M < D$.

Let:

$$B = YY^T = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \begin{bmatrix} y_1^T & y_2^T & \cdots & y_N^T \end{bmatrix} = \begin{bmatrix} y_1 y_1^T & y_1 y_2^T & \cdots & y_1 y_N^T \\ y_2 y_1^T & y_2 y_2^T & \cdots & y_2 y_N^T \\ \vdots & \vdots & \ddots & \vdots \\ y_N y_1^T & y_N y_2^T & \cdots & y_N y_N^T \end{bmatrix}$$

Then

$$b_{ij} = \sum_{k=1}^{M} y_{ik} y_{jk}$$

- So, if we want to find B, then we can determine Y by taking:

$$Y \approx B^{1/2}$$

since $B = YY^T$.

- Last class, we saw that we can write the matrix $\mathbf{B}$ using the proximity matrix D:

$$b_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{N}\sum_{j=1}^{N} d_{ij}^2 - \frac{1}{N}\sum_{i=1}^{N} d_{ij}^2 + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}^2 \right)$$

- So, now, we can estimate B using the proximity matrix D. In a matrix form, we can write:

$$B = -\frac{1}{2} J D^2 J$$

where

$$J = I - \frac{1}{N}\mathbf{1}\mathbf{1}^T$$

and I is an $N \times N$ identity matrix, $\mathbf{1}$ is an $N \times 1$ vector of 1's and $D^2 = \left[d_{ij}^2\right]$ is the proximity matrix of size $N \times N$.

- Recall that for any real symmetruc matrix B, the eigenvalues are real and the eigenvectors can be chosen such that they are orthogonal to each other. Thus a real symmetric matrix B can be described in the eigenspace, as follows:

$$B = V \Lambda V^T$$

where V is an orthogonal matrix containing the eigenvectors of B and Λ is a diagonal matrix containing the eigenvalues of B.

Then we can estimate Y as follows:

$$Y = B^{1/2} = V \Lambda^{1/2}$$

keeping only the M dimensions of interest ($M < D$) corresponding to the M largest eigenvalues.

- Note: If we use the Euclidean distance to compute D then MDS is equivalent to PCA! However, D can be computed with any metric and therefore, MDS is a generalization of PCA.

Steps to Implement MDS

The MDS algorithm:

1. Compute the distance/proximity matrix, D.
2. Compute D^2.
3. Compute $J = I - \frac{1}{N}\mathbf{1}\mathbf{1}^T$
4. Compute $B = -\frac{1}{2} J D^2 J$
5. Compute eigenvectors, V, and eigenvalues,Λ, of B. (store them in a matrix in decreasing eigenvalue order.)
6. Compute $Y = V \Lambda^{1/2}$

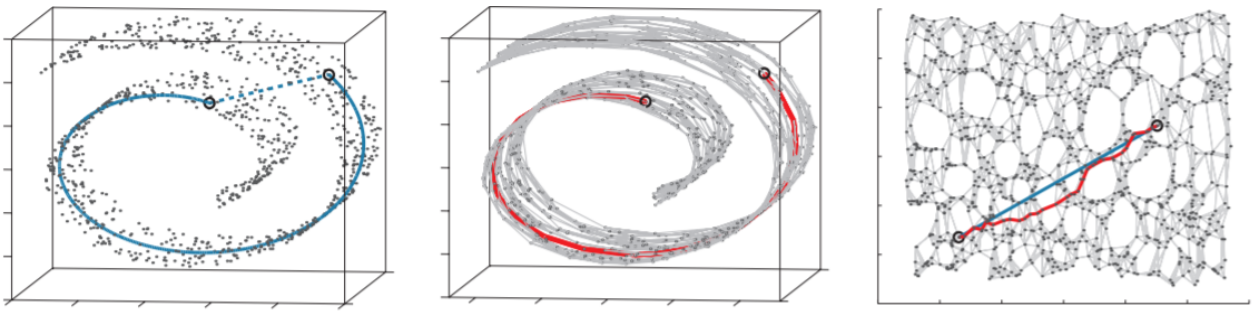
Isometric feature Mapping (ISOMAP)

In **Isometric feature Mapping**, or **ISOMAP**, the goal is to project the data to a lower-dimensional space using MDS, but where the distance/dissimilarities are defined in terms of the geodesic distances measured along the manifold.

- *How can we address the same problem when the data does not lie on a linear subspace but rather on a sub-manifold?*

In [5]:
```
Image('figures/isomap.png', width=900)
```

Out[5]:



ISOMAP was introduced in 2000 in a Science issue paper and approaches dimensionality reduction for data that lies on a manifold.

- It builds on classical MDS.
- The contribution of ISOMAP is that the proximity matrix, D, is constructed using **geodesic distances**.
- Each data point and its neighbors lie on or close to a locally linear patch of the manifold. And, so, each data point can be reconstructed as a weighted sum of its neighbors.

Steps to Implement ISOMAP

1. **Construct neighborhood graph:** Identify the neighbors of each point on the manifold (e.g., K nearest neighbors, ϵ-ball approach). The edges between neighbors are weighted using their distance in the input space.
2. **Compute shortest paths:** Compute the distance between all pairs of points on the manifold by computing their shortest path distance on the graph created in step 1 (e.g. Floyd-Warshall's algorithm). This will produce the distance matrix D_G.
3. **Construct M-dimensional embedding:** Apply classical MDS to the matrix of graph distances computed in step 2

$$B = -\frac{1}{2} J D_G^2 J$$

$$Y = V \Lambda^{1/2}$$

where V and Λ are the eigenvectors and eigenvalues of B, respectively.

- More details on the mathematical derivations can be found in the original published work (here).

Floyd-Warshall Algorithm

The Foyd-Warshall alorithm to compute graph-based distance:

1. Initialize $d_G(i,j) = d_X(i,j)$ if i, j are identified as neighbors. Otherwise, set $d_G(i,j) = \infty$.

    - This means that, if data points are considered neighbors of a particular point, then Euclidean distance between that point and its neighbors is sufficient and assumed to be close enough to the geodesic distance. (Note that this is true for a reasonably small number of neighbors.)
2. For each value $k = 1, 2, \ldots, N$, replace all entries of $d_G(i, j)$ by $\min\left(d_G(i, j), d_G(i, k) + d_G(k, j)\right)$.

The matrix of final values $D_G = \{d_G(i, j)\}$ will contain the **shortest path distances** between all pairs of points in G.

Shortcomings of ISOMAP

If the data matrix $X \in \mathbb{R}^{N \times D}$ is sufficiently dense, then graph shortest path distance will approximate closely the original geodesic distance.

- ISOMAP may suffer from *non-convexity* such as holes on manifols.
- We need to compute pairwise shortest distance path between **all** sample pairs (i, j). This matrix is a global matrix, non-sparse and it requires cubic complexity $O(N^3)$.

Locally Linear Embedding (LLE)

Locally linear embedding, or **LLE** first computes the set of coefficients that best reconstructs each data point from its neighbors. These coefficients are arranged to be invariant to rotations, translations, and scalings of that data point and its neighbors, and hence they characterize the local geometrical properties of the neighborhood.

- LLE was also introduced in 2000 in the same Science issue as ISOMAP.
- The intuition behind LLE is that each data point and its close neighbors lie on or close to a *locally linear* patch of the manifold.
    - We can characterize the local geometry of these patches by linear coefficients that reconstruct each data point from its neighbors.
- Each point can be written as a linear combination of its neighbors.
- The lower-dimensional projection will be a combination of locally linear patches.

Informally, imagine taking a pair of scissors, cutting out locally linear patches of the underlying manifold, and placing them in the low dimensional embedding space. Assume further that this operation is done in a way that preserves the angles formed by each data point to its nearest neighbors. In this case, the transplantation of each patch involves no more than a translation, rotation, and rescaling of its data, exactly the operations to which the weights are invariant. Thus, when the patch arrives at its low dimensional destination, we expect the same weights to reconstruct each data point from its neighbors.

Steps to Implement LLE

1. **Construct neighborhood graph:** Identify the neighbors of each point on the manifold (e.g., K nearest neighbors, ϵ-ball approach). The edges between neighbors are weighted using their distance in the input space.

2. Find a set of weights $W \in \mathbb{R}^{D \times K}$ such that each point $x_i \in \mathbb{R}^{D \times 1}$ can be reconstructed by its K neighbors.

$$x_i = \sum_{j=1}^{K} w_{ij} x_{i(j)}$$

where $x_{i(j)}$ is the j-th neighbor of x_i.

The reconstruction errors are measured by the cost function:

$$\epsilon(W) = \sum_{i=1}^{N} \left\| x_i - \sum_{j=1}^{K} w_{ij} x_{i(j)} \right\|_2^2$$

which adds up the squared distances between all the data points and their reconstructions. The weights w_{ij} summarize the contribution of the j-th data point to the i-th reconstruction. To compute the weights w_{ij}, we minimize the cost function $\epsilon(W)$ subject to two constraints:

Contraint 1: Each data point x_i is reconstructed only from its neighbors, enforcing $w_{ij} = 0$ if x_j is not a neighbor of x_i.

Constraint 2: The rows of the weight matrix sum to one: $\sum_{j=1}^{K} w_{ij} = 1$. The optimal weights subject to these constraints are found by solving a least-squares problem.

1. Each high-dimensional observation $x_i \in \mathbb{R}^{D \times 1}$ is mapped to a low-dimensional vector $x_i \in \mathbb{R}^{M \times 1}$ representing global internal coordinates on the manifold. This is done by choosing M-dimensional coordinates y_i to minimize the embedding cost function:

$$\Phi(Y) = \sum_{i=1}^{N} \left\| y_i - \sum_{j=1}^{K} w_{ij} y_j \right\|_2^2$$

with fixed weights W. To compute the projection Y, we minimize the cost function $\Phi(Y)$ subject to two constraints:

Contraint 1: The coordinates are centered on the origin: $\sum_{i=1}^{N} y_i = 0$

Contraint 2: Also, to avoid degenerate solutions, we constrain the embedding vectors to have unit covariance, with outer products that satisfy $\frac{1}{N} Y Y^T = I$

- More details on the mathematical derivations can be found in the original published work (here).

Step 2: Reconstruction of Errors

For each data point $\mathbf{x}_i \in \mathbb{R}^{D\times 1}$ define its matrix of K neighbors to be

$$\mathbf{V}_i = \begin{bmatrix} x_{i(1)} & x_{i(2)} & \cdots & x_{i(k)} \end{bmatrix}$$

where $\mathbf{V}_i$ is a $D \times K$ matrix. And define the weight vector as:

$$\mathbf{w}_i = \begin{bmatrix} w_{i1} & w_{i2} & \cdots & w_{ik} \end{bmatrix}^T$$

where $\mathbf{w}_i$ is a $K \times 1$ vector.

The reconstruction of data point $\mathbf{x}_i$ can be written as:

$$\epsilon(\mathbf{w}_i) = \left\| \mathbf{x}_i - \mathbf{V}_i \mathbf{w}_i \right\|^2$$

Let $\mathbf{1} = [1, 1, \ldots, 1]^T$ a $K \times 1$ vector. Construct a matrix of the form $[\mathbf{x}_i, \mathbf{x}_i, \ldots, \mathbf{x}_i]_{D\times K} = \mathbf{x}_i \mathbf{1}^T$. We can then write:

$$\mathbf{x}_i = \mathbf{x}_i \mathbf{1}^T \mathbf{w}_i$$

because $\sum_{j=1}^{K} \mathbf{w}_{ij} = 1$.

Rewriting the errors function:

$$\begin{aligned}
\epsilon(\mathbf{w}_i) &= \left\| \mathbf{x}_i - \mathbf{V}_i \mathbf{w}_i \right\|^2 && (1)\\
&= \left\| \mathbf{x}_i \mathbf{1}^T \mathbf{w}_i - \mathbf{V}_i \mathbf{w}_i \right\|^2 && (2)\\
&= \left\| \left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right) \mathbf{w}_i \right\|^2 && (3)\\
&= \left(\left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right) \mathbf{w}_i \right)^T \left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right) \mathbf{w}_i && (4)\\
&= \mathbf{w}_i^T \left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right)^T \left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right) \mathbf{w}_i && (5)\\
&= \mathbf{w}_i^T \mathbf{G} \mathbf{w}_i && (6)
\end{aligned}$$

where $\mathbf{G} = \left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right)^T \left(\mathbf{x}_i \mathbf{1}^T - \mathbf{V}_i \right)$ is a known $K \times K$ matrix and is called the *Gram* matrix.

To find the weights, we solve the following optimization:

$$\min_{\mathbf{w}_i} \mathbf{w}_i^T \mathbf{G} \mathbf{w}_i \text{ such that } \sum_{j=1}^{K} \mathbf{w}_{ij} = \mathbf{w}_i^T \mathbf{1} = 1$$

We can solve this optimization using Lagrangian optimization with linear equality constraints:

$$\mathcal{L} = \mathbf{w}_i^T \mathbf{G} \mathbf{w}_i + \lambda (1 - \mathbf{w}_i^T \mathbf{1})$$

Then,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} &= 0 && (7)\\
2\mathbf{G}\mathbf{w}_i - \lambda \mathbf{1} &= 0 && (8)\\
\mathbf{w}_i &= \frac{\lambda}{2} \mathbf{G}^{-1} \mathbf{1} && (9)
\end{aligned}$$

We can solve for any arbitrary λ and then normalize $\mathbf{w}_i$ such that its sum is equal to 1.

$$\mathbf{w}_i = \frac{\mathbf{w}_i}{\sum_{j=1}^{K} w_{ij}} \tag{10}$$

$$= \begin{bmatrix} \frac{\mathbf{w}_{i1}}{\sum_{j=1}^{K} w_{ij}} & \frac{\mathbf{w}_{i2}}{\sum_{j=1}^{K} w_{ij}} & \cdots & \frac{\mathbf{w}_{ik}}{\sum_{j=1}^{K} w_{ij}} \end{bmatrix}^T \tag{11}$$

Note that:

- Different constraints can be employed on the weights $\mathbf{w}_i$
- The Gram matrix $\mathbf{G}$ is not be invertible if the number of neighbors K is larger than the number of dimensions D. For this case, we can diagonally load the Gram matrix $\mathbf{G}$ and then invert it

This completes step 2 of LLE algorithm.

Step 3: Finding low-dimensional transformation

For the third and final step of LLE, we seek to find a lower-dimensional transformation $\mathbf{Y} \in \mathbb{R}^{d \times N}$ ($d < D$) using the weights we found in the previous step. To do so, we need to optimize the objective function:

$$\Phi(\mathbf{Y}) = \sum_{i=1}^{N} \left\| y_i - \sum_{j=1}^{K} w_{ij} y_j \right\|^2 \tag{12}$$

$$\text{where } \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_N \end{bmatrix}_{d \times N} \tag{13}$$

Let $\mathbf{I}_{N \times N} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{N \times N}$ and $I_{c_i} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{N \times 1}$ the i-th column of the identity matrix. I_{c_i} is a $N \times 1$ vector with 1 at the i-th position and 0 everywhere else.

Now consider the matrix of weights $\mathbf{W}$ of size $N \times N$ where the k neighbors have some weight value and zeros for any other points that are not one of the k neighbors. So we can write one of the columns of $\mathbf{W}$ as:

$$\mathbf{W}_{c_i} = \begin{bmatrix} 0 \\ 0 \\ w_{i1} \\ w_{i2} \\ \vdots \\ w_{ik} \\ 0 \end{bmatrix}_{N \times 1}$$

where $w_{i1}, w_{i2}, \ldots, w_{ik}$ are k-nearest neighbors weights of x_i.

Using these notations, we can write:

$$\mathbf{y}_i = \mathbf{Y}\mathbf{I}_{c_i} \tag{14}$$
$$\sum_{j=1} w_{ij} y_j = \mathbf{Y}\mathbf{w}_{c_i} \tag{15}$$

We are now optimizing a cost function of $\mathbf{Y}$ with known weight vector $\mathbf{W}$:

$$\begin{aligned}
\min_{\mathbf{Y}} \sum_{i=1}^{N} \left\| y_i - \sum_{j=1}^{K} w_{ij} y_j \right\|^2 &= \min_{\mathbf{Y}} \sum_{i=1}^{N} \left\| \mathbf{Y}\mathbf{I}_{c_i} - \mathbf{Y}\mathbf{w}_{c_i} \right\|^2 && (16\\
&= \min_{\mathbf{Y}} \left\| \mathbf{Y}\mathbf{I} - \mathbf{Y}\mathbf{W} \right\|^2 && (17\\
&= \min_{\mathbf{Y}} \left\| \mathbf{Y}\left(\mathbf{I} - \mathbf{W}\right) \right\|^2 && (18\\
&= \min_{\mathbf{Y}} Tr\left(\left(\mathbf{Y}\left(\mathbf{I} - \mathbf{W}\right)\right)^T \mathbf{Y}\left(\mathbf{I} - \mathbf{W}\right) \right) && (19\\
&= \min_{\mathbf{Y}} Tr\left(\left(\mathbf{I} - \mathbf{W}\right)^T \mathbf{Y}^T \mathbf{Y}\left(\mathbf{I} - \mathbf{W}\right) \right) && (20\\
&= \min_{\mathbf{Y}} Tr\left(\left(\mathbf{I} - \mathbf{W}\right)\left(\mathbf{I} - \mathbf{W}\right)^T \mathbf{Y}^T \mathbf{Y} \right), \text{because trace is cyclic} && (21\\
&= \min_{\mathbf{Y}} Tr\left(\mathbf{Y}\left(\mathbf{I} - \mathbf{W}\right)\left(\mathbf{I} - \mathbf{W}\right)^T \mathbf{Y}^T \right) && (22\\
&= \min_{\mathbf{Y}} Tr\left(\mathbf{Y}\mathbf{M}\mathbf{Y}^T \right) && (23
\end{aligned}$$

where $\mathbf{M} = \left(\mathbf{I} - \mathbf{W}\right)\left(\mathbf{I} - \mathbf{W}\right)^T$.

We have the constraints $\sum_{i=1}^{N} \mathbf{y}_i = 0$ and $\frac{1}{N}\mathbf{Y}\mathbf{Y}^T = \mathbf{I}_{N\times N}$

$$\min_{\mathbf{Y}} Tr\left(\mathbf{Y}\mathbf{M}\mathbf{Y}^T \right) \text{ such that } \frac{1}{N}\mathbf{Y}\mathbf{Y}^T = \mathbf{I}_{N\times N} \text{ and } \sum_{i=1}^{N} \mathbf{y}_i = 0$$

This is a classical problem regarding trace optimization with a known solution: the lower dimensional projection coordinates correspond to the d eigenvectors of $\mathbf{M}$ corresponding to the d \textbf{smallest} eigenvalues different than 0.

Properties of $\mathbf{M}$:

- $\mathbf{M}$ is *positive semi-definite* meaning that ALL eigenvalues are greater or equal to zero
- $\mathbf{M}$ always has one eigenvalue equal to 0 with corresponding eigenvector equal to $[1, 1, \ldots, 1]^T$ which is not useful for our problem so we ignore this eigenvector
- In fact, we need to pick the $d + 1$ eigenvectors corresponding to the \textbf{smallest} eigenvalues

Consider the eigenvalues of $\mathbf{M}$ sorted (in decreading order) as $\lambda_1 < \lambda_2 < \cdots < \lambda_N$ with corresponding eigenvectors $\{\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_N\}$

- **Solution:** $\mathbf{Y}$ contains the d eigenvectors of $\mathbf{M}$ corresponding to the d \textbf{smallest} eigenvalues different than 0 in \textbf{rows}

$$\mathbf{Y} = \begin{bmatrix} \mathbf{e}_2 \\ \mathbf{e}_3 \\ \vdots \\ \mathbf{e}_{d+1} \end{bmatrix}_{d \times N}$$

Examples

Let's run the .py files to see some examples.