Lecture 9: Dimension Reduction and Data Embedding Techniques

Radu Balan

Department of Mathematics, AMSC, CSCAMM and NWC
University of Maryland, College Park, MD

April 22 and 29, 2021

Input Data

Available Information:

1. Geometric Graph: For a threshold $\tau \geq 0$, $\mathcal{G}_\tau = (\mathcal{V}, \mathcal{E}, \mu)$ where $\mathcal{V}$ is the set of n vertices (nodes), $\mathcal{E}$ is the set of edges between nodes i and j if $\|x_i - x_j\| \leq \tau$ and $\mu : \mathcal{E} \to \mathbb{R}$ the set of distances $\|x_i - x_j\|$ between nodes connected by en edge.
2. Weighted graph: $\mathcal{G} = (\mathcal{V}, W)$ a undirected weighted graph with n nodes and weight matrix W, where $W_{i,j}$ is inverse monotonically dependent to distances $\|x_i - x_j\|$; the smaller the distance $\|x_i - x_j\|$ the larger the weight $W_{i,j}$.
3. Unweighted graph: For a threshold $\tau \geq 0$, $\mathcal{U}_\tau = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of n nodes, and $\mathcal{E}$ is the set of edges connected node i to node j if $\|x_i - x_j\| \leq \tau$. Note the distance information is not available.

Thus we look for a dimension $d > 0$ and a set of points $\{y_1, y_2, \cdots, y_n\} \subset \mathbb{R}^d$ so that all $d_{i,j} = \|y_i - y_j\|$'s are *compatible* with input data as defined above.

Approaches

Popular Approaches:

1. Principal Component Analysis
2. Independent Component Analysis
3. Laplacian Eigenmaps
4. Local Linear Embeddings (LLE)
5. Isomaps

If points were supposed to belong to a lower dimensional manifold, the problem is known under the term *manifold learning*. If the manifold is linear (affine), then the Principal Component Analysis (PCA) or Independent Component Analysis (ICA) would suffice. However, if the manifold is not linear, then nonlinear methods are needed. In this respect, Laplacian Eigenmaps, LLE and ISOMAP can be thought of as *nonlinear PCA*. Also known as *nonlinear embeddings*.

Principal Component Analysis

Approach

Data: We are given a set $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^N$ of n points in $\mathbb{R}^N$.
Goal: We want to find a linear (or affine) subspace V of dimension d that best approximates this set. Specifically, if $P = P_V$ denotes the orthogonal projection onto V, then the goal is to minimize

$$J(V) = \sum_{k=1}^{n} \|x_k - P_V x_k\|_2^2.$$

If V is linear space (i.e. passes through the origin) then P is $N \times N$ linear operator (i.e. matrix) that satisfies $P = P^T$, $P^2 = P$, and $Ran(P) = V$. If V is an affine space (i.e. a linear space shifted by a constant vector), then the projection onto the affine space is $T(x) = Px + b$ where b is a constant vector (the "shift").
The affine space case can be easily reduced to the linear space: just append 1 to the bottom of each vector x_k: $\tilde{x}_k = [x_k; 1]$. Now b becomes a column of the extended matrix $\tilde{P} = [P\ b]$

Principal Component Analysis

Algorithm

Algorithm (Principal Component Analysis)

Input: Data vectors $\{x_1, \cdots, x_n\} \in \mathbb{R}^N$*; dimension* d.

0. *If affine subspace is the goal, append '1' at the end of each data vector.*
1. *Compute the sample covariance matrix*

$$R = \sum_{k=1}^{n} x_k x_k^T$$

2. *Solve the eigenproblems* $Re_k = \sigma_k^2 e_k$, $1 \leq k \leq N$, *order eigenvalues* $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_N^2$ *and normalize the eigenvectors* $\|e_k\|_2 = 1$.

Principal Component Analysis

Algorithm - cont'ed

Algorithm (Principal Component Analysis)

3. *Construct the co-isometry*

$$U = \begin{bmatrix} e_1^T \\ \vdots \\ e_d^T \end{bmatrix}.$$

4. *Project the input data*

$$y_1 = Ux_1 \ , \ y_2 = Ux_2 \ , \ \cdots \ , \ y_n = Ux_n.$$

Output: Lower dimensional data vectors $\{y_1, \cdots, y_n\} \in \mathbb{R}^d$.

The orthogonal projection is given by $P = \sum_{k=1}^{d} e_k e_k^T$ and the optimal subspace is $V = Ran(P)$.

Principal Component Analysis

Derivation

Here is the derivation in the case of linear space. The reduced dimensional data is given by Px_k. Expand the criterion $J(V)$:

$$J(V)=\sum_{k=1}^{n}\|x_k\|^2-\sum_{k=1}^{n}\langle Px_k,x_k\rangle=\sum_{k=1}^{n}\|x_k\|^2-trace(PR)$$

where $R=\sum_{k=1}^{n}x_kx_k^T$. It follows the minimizer of $J(V)$ maximizes $trace(PR)$ subject to $P=P^T$, $P^2=P$ and $trace(P)=d$. It follows the optimal P is given by the orthogonal projection onto the top d eigenvectors, hence the algorithm.

Independent Component Analysis

Approach

Model (Setup): $x = As$, where A is an unknown invertible $N \times N$ matrix, and $s \in \mathbb{R}^N$ is a random vector of *independent* components.
Data: We are given a set of measurement $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^N$ of n points in $\mathbb{R}^N$ of the model $x_k = As_k$, where each $\{s_1, \cdots, s_n\}$ is drawn from the same distribution $p_s(s)$ of N-vectors with independent components.
Goal: We want to estimate the invertible matrix A and the (source) signals $\{s_1, \cdots, s_n\}$. Specifically, we want a square matrix W such that Wx has independent components.
Principle: Perform PCA first so the decorrelated signals have unit variance. Then find an orthogonal matrix (that is guaranteed to preserve decorrelation) that creates statistical independence as much as possible.
Caveat: Two inherent ambiguities: (1) Permutation: If W is a solution to the unmixing problem so is ΠW, where Π is a permutation matrix; (2) Scaling: If W is a solution to unmixing problem, so is DW where D is a diagonal matrix.

Independent Component Analysis

Algorithm

Algorithm (Independent Component Analysis)

Input: *Data vectors* $\{x_1, \cdots, x_n\} \in \mathbb{R}^N$.

1. *Compute the sample mean* $b = \frac{1}{n}\sum_{k=1}^{n} x_k$, *and sample covariance matrix* $R = \frac{1}{n}\sum_{k=1}^{n}(x_k - b)(x_k - b)^T$.
2. *Solve the eigenproblem* $RE = E\Lambda$, *where* E *is the* $N \times N$ *orthogonal matrix whose columns are eigenvectors, and* Λ *is the diagonal matrix of eigenvalues.*
3. *Compute* $F = R^{-1/2} := E\Lambda^{-1/2}E^T$ *and apply it on data,* $z_k = F(x_k - b)$, $1 \leq k \leq n$.
4. *Compute the orthogonal matrix* Q *using the JADE algorithm below.*
5. *Apply* Q *on whitened data,* $\hat{s}_k = Qz_k$, $1 \leq k \leq n$. *Compute* $W = QF$.

Output: *Matrix* W *and independent vectors* $\{\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_n\}$.

Independent Component Analysis – Cont.

Joint Approximate Diagonalization of Eigenmatrices (JADE)

Algorithm (Cardoso's 4th Order Cumulants Algorithm'92)

Input: Whitened data vectors $\{z_1, \cdots, z_n\} \in \mathbb{R}^N$.

1. *Compute the sample* 4^{th} *order symmetric cumulant tensor*

$$F_{ijkl} = \frac{1}{N}\sum_{t=1}^{N} z_t(i)z_t(j)z_t(k)z_t(l) - \delta_{i,j}\delta_{k,l} - \delta_{i,k}\delta_{j,l} - \delta_{i,l}\delta_{j,k}.$$

2. *Compute* N *eigenmatrices* $M_{i,j}$*, so that* $F(M_{i,j}) = \lambda_{i,j}M_{i,j}$.
3. *Maximize the criterion*

$$J_{JADE}(Q) = \sum_{i,j} |\lambda_{i,j}|^2 \|diag(QM_{i,j}Q^T)\|_2^2$$

over orthogonal matrices Q *by performing successive rotations marching through all pairs* (a, b) *of distinct indices in* $\{1, \cdots, N\}$.

Independent Component Analysis – Cont.

Wang-Amari Natural Stochastic Gradient Algorithm of Bell-Sejnowski MaxEntropy

Algorithm (Wang-Amari'97; Bell-Sejnowski'95)

Input: Sphered data vectors $\{z_1, \cdots, z_n\} \in \mathbb{R}^N$*; Cumulative distribution functions* g_k *of each component of* s*; Learning rate* η*.*

1. *Initialize* $W^{(0)} = F$.
2. *Repeat until convergence, or until maximum number of steps reached:*
    1. *Draw a data vector* z *randomly from data vectors, and compute*

       $$W^{(t+1)} = W^{(t)} + \eta(I + (1 - 2g(z))z^T)W^{(t)}.$$

    2. *increment* $t \leftarrow t + 1$.

Output: Unmixing $N \times N$ *matrix* $W = W^{(T)}$.

Independent Component Analysis

Derivation

Dimension Reduction using Laplacian Eigenmaps

Idea

First, convert any relevant data into an undirected weighted graph, hence a symmetric weight matrix W.
Assume $\mathcal{G} = (\mathcal{V}, W)$ is a undirected weighted graph with n nodes and weight matrix W.
We interpret $W_{i,j}$ as the *similarity* between nodes i and j. The larger the weight the more similar the nodes, and the closer they are in a geometric graph embedding.
Thus we look for a dimension $d > 0$ and a set of points $\{y_1, y_2, \cdots, y_n\} \subset \mathbb{R}^d$ so that $d_{i,j} = \|y_i - y_j\|$'s is small for large weight $W_{i,j}$.

Dimension Reduction using Laplacian Eigenmaps

Idea

First, convert any relevant data into an undirected weighted graph, hence a symmetric weight matrix W.

Assume $\mathcal{G} = (\mathcal{V}, W)$ is a undirected weighted graph with n nodes and weight matrix W.

We interpret $W_{i,j}$ as the *similarity* between nodes i and j. The larger the weight the more similar the nodes, and the closer they are in a geometric graph embedding.

Thus we look for a dimension $d > 0$ and a set of points $\{y_1, y_2, \cdots, y_n\} \subset \mathbb{R}^d$ so that $d_{i,j} = \|y_i - y_j\|$'s is small for large weight $W_{i,j}$.

A natural optimization criterion candidate:

$$J(y_1, y_2, \cdots, y_n) = \frac{1}{2} \sum_{1 \le i,j \le n} W_{i,j} \|y_i - y_j\|^2,$$

Optimization Criteria

Lemma

$$J(y_1, y_2, \cdots, y_n) = \frac{1}{2} \sum_{1 \leq i,j \leq n} W_{i,j} \|y_i - y_j\|^2$$

is convex in $(y_1, \cdots, y_n)$.

Proof Idea: Write it as a positive semidefinite quadratic criterion:

$$J = \sum_{i=1}^{n} \|y_i\|^2 \sum_{j=1}^{n} W_{i,j} - \sum_{i,j=1}^{n} W_{i,j} \langle y_i, y_j \rangle$$

Let $Y = [y_1 | \cdots | y_n]$ be the $d \times n$ matrix of coordinates. Let $D = diag(d_k)$, with $d_k = \sum_{i=1}^{n} W_{k,i}$, be a $n \times n$ diagonal matrix. A little algebra shows:

$$J(Y) = trace\left\{ Y(D - W)Y^T \right\}.$$

Optimization Criteria

Equivalent forms:

$$J(Y) = trace\left\{Y(D-W)Y^T\right\} = trace\left\{Y\Delta Y^T\right\} = trace\left\{\Delta G\right\}$$

where $G = Y^T Y$ is the $n \times n$ Gram matrix. Thus: J is quadratic in Y, and positive semidefinite, hence convex.
Also: J is linear in G.
Question: Are there other convex functions in Y that behave similarly?

Optimization Criteria

Equivalent forms:

$$J(Y) = trace\left\{Y(D-W)Y^T\right\} = trace\left\{Y\Delta Y^T\right\} = trace\left\{\Delta G\right\}$$

where $G = Y^T Y$ is the $n \times n$ Gram matrix. Thus: J is quadratic in Y, and positive semidefinite, hence convex.
Also: J is linear in G.
Question: Are there other convex functions in Y that behave similarly?
Answer: Yes! Examples:

$$J(y_1, \cdots, y_n) = \sum_{1 \le i,j \le n} W_{i,j} \|y_i - y_j\|$$

$$J(y_1, \cdots, y_n) = \left(\sum_{1 \le i,j \le n} W_{i,j} \|y_i - y_j\|^p\right)^{1/p} \quad , \quad p \ge 1$$

Constraints

Absent any constraint,

$$minimize \quad trace\left\{Y\Delta Y^T\right\}$$

has solution $Y = 0$. To avoid this trivial solution, we impose a normalization constraint.
Choices:

$$YY^T = I_d$$
$$YDY^T = I_d$$

What does this mean?

$$\sum_{k=1}^{n} y_k y_k^T = I_d \quad \Rightarrow \quad \text{Parseval frame}$$

$$\sum_{k=1}^{n} d_k y_k y_k^T = I_d \quad \Rightarrow \quad \text{Parseval weighted frame}$$

The Optimization Problem

Combining one criterion with one constraint:

$$(LE) \quad : \quad \begin{array}{ll} \text{minimize} & trace\left\{Y \Delta Y^T\right\} \\ \text{subject to} & YDY^T = I_d \end{array}$$

called the *Laplacian Eigenmap* problem.

Alternative problem:

$$(UnLE) \quad : \quad \begin{array}{ll} \text{minimize} & trace\left\{Y \Delta Y^T\right\} \\ \text{subject to} & YY^T = I_d \end{array}$$

called the *unnormalized Laplacian eigenmap* problem.

The optimization problem

How to solve the Laplacian eigenmap problem:

$$(LE) \; : \quad \begin{array}{ll} \text{minimize} & trace\left\{Y \Delta Y^T\right\} \\ \text{subject to} & YDY^T = I_d \end{array}$$

First note the problem is not convex, because of the equality constraint. How to make it convex? How to solve?

1. First absorb the scaling D into the solution:

$$\tilde{Y} = YD^{1/2}$$

Problem becomes:

$$\begin{array}{ll} \text{minimize} & trace\left\{\tilde{Y} D^{-1/2} \Delta D^{-1/2} \tilde{Y}^T\right\} = trace\left\{\tilde{Y} \tilde{\Delta} \tilde{Y}^T\right\} \\ \text{subject to} & \tilde{Y} \tilde{Y}^T = I_d \end{array}$$

The optimization problem

2. Consider the optimization problem for P:

$$\begin{aligned} \text{minimize} \quad & trace\left\{\tilde{\Delta} P\right\} \\ \text{subject to} \quad & P = P^T \geq 0 \\ & P \leq I_n \\ & trace(P) = d \end{aligned}$$

Proposition

With notations above:
A. The above optimization problem is a convex SDP.
B. At optimum: $P = \tilde{Y}^T \tilde{Y}$.

The optimization problem

Why?

Eigenproblem

The optimum solutions of the (LE) and (UnLE) problems are given by appropriate eigenvectors:

$$\begin{array}{ll} \text{minimize} & trace\left\{\tilde{Y}\tilde{\Delta}\tilde{Y}^T\right\} \\ \text{subject to} & \tilde{Y}\tilde{Y}^T = I_d \end{array}$$

Solution:

$$\tilde{Y} = \begin{bmatrix} e_1^T \\ \vdots \\ e_d^T \end{bmatrix} \quad , \quad \tilde{\Delta} e_k = \lambda_k e_k$$

where $0 = \lambda_1 \leq \cdots \lambda_d$ are the smallest d eigenvalues, and $\|e_k\| = 1$ are the normalized eigenvectors.

Generalized Eigenproblem

$$(LE) \; : \quad \begin{array}{ll} \text{minimize} & trace\left\{Y \Delta Y^T\right\} \\ \text{subject to} & YDY^T = I_d \end{array} \quad \Rightarrow \quad Y = \tilde{Y} D^{-1/2}$$

the rows of $\tilde{Y}$ are eigenvectors of the normalized Laplacian $\tilde{\Delta} e_k = \lambda_k e_k$. Let f_k be the (transpose) rows of Y:

$$Y = \begin{bmatrix} f_1^T \\ \vdots \\ f_d^T \end{bmatrix} \quad , \quad f_k = D^{-1/2} e_k$$

Thus: $\tilde{\Delta} D^{1/2} f_k = \lambda_k D^{1/2} f_k$, or: $D^{1/2} \tilde{\Delta} D^{1/2} f_k = \lambda_k D f_k$, or:

$$\Delta f_k = \lambda_k D f_k$$

This is called *generalized eigenproblem* associated to (Δ, D).

Eigenproblem

Consider the unnormalized Laplacian eigenmap problem:

$$(UnLE) \; : \; \begin{array}{ll} \text{minimize} & trace\left\{Y\Delta Y^T\right\} \\ \text{subject to} & YY^T = I_d \end{array} .$$

The solution Y^{unLE} is the $d \times n$ matrix whose rows are eigenvectors of the unnormalized Laplacian $\Delta = D - W$, $\Delta g_k = \mu_k g_k$, $\|g_k\| = 1$, $0 = \mu_1 \leq \cdots \leq \mu_d$, and

$$Y^{unLE} = \begin{bmatrix} g_1^T \\ \vdots \\ g_d^T \end{bmatrix} .$$

What eigenspace to choose?

In most implementations one skips the eigenvectors associated to 0 eigenvalue. Why? In the unnormalized case, $g_1 = \frac{1}{\sqrt{n}}[1, 1, \cdots, 1]^T$, hence no new information.

In your class projects, skip the bottom eigenvector.

Laplacian Eigenmaps Embedding

Algorithm

Algorithm (Laplacian Eigenmaps)

Input: Weight matrix W, target dimension d

1. *Construct the diagonal matrix* $D = diag(D_{ii})_{1\leq i\leq n}$, *where* $D_{ii} = \sum_{k=1}^{n} W_{i,k}$.
2. *Construct the normalized Laplacian* $\tilde{\Delta} = I - D^{-1/2} W D^{-1/2}$.
3. *Compute the bottom* $d+1$ *eigenvectors* $e_1, \cdots, e_{d+1}$, $\tilde{\Delta} e_k = \lambda_k e_k$, $0 = \lambda_1 \leq \cdots \leq \lambda_{d+1}$.

Laplacian Eigenmaps Embedding

Algorithm-cont's

Algorithm (Laplacian Eigenmaps - cont'd)

4. *Construct the* $d \times n$ *matrix* Y,

$$Y = \begin{bmatrix} e_2^T \\ \vdots \\ e_{d+1}^T \end{bmatrix} D^{-1/2}$$

5. *The new geometric graph is obtained by converting the columns of* Y *into* n d*-dimensional vectors:*

$$\begin{bmatrix} y_1 & | & \cdots & | & y_n \end{bmatrix} = Y$$

Output: Set of points $\{y_1, y_2, \cdots, y_n\} \subset \mathbb{R}^d$.

Dimension Reduction using Laplacian Eigenmaps

Idea

Given a high dimensional geometric graph $\{x_1, \cdots, x_n\} \subset \mathbb{R}^N$ we seek a lower dimensional geometric set of points $\{y_1, \cdots, y_n\} \subset \mathbb{R}^d$ whose pairwise distances are compatible with the original graph.
First, convert input data into an undirected weighted graph, hence a symmetric weight matrix W. For instance, for each pair $1 \leq i < j \leq n$,

$$W_{i,j} = exp(-\alpha \|x_i - x_j\|^2) \ , \ if \|x_i - x_j\| \leq \tau \quad , \quad W_{i,j} = 0 \ , \ otherwise$$

The Laplacian eigenmaps solve the following optimization problem:

$$(LE) \ : \ \begin{array}{ll} \text{minimize} & trace\left\{ Y \Delta Y^T \right\} \\ \text{subject to} & YDY^T = I_d \end{array}$$

where $\Delta = D - W$ with D the diagonal matrix $D_{ii} = \sum_{k=1}^{n} W_{i,k}$
The $d \times n$ matrix $Y = [y_1 | \cdots | y_n]$ contains the embedding.

Dimension Reduction using Laplacian Eigenmaps

Algorithm

Algorithm (Dimension Reduction using Laplacian Eigenmaps)

Input: A geometric graph $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^N$. *Parameters: threshold* τ, *weight coefficient* α, *and dimension* d.

1. *Compute the set of pairwise distances* $\|x_i - x_j\|$ *and convert them into a set of weights:*

$$W_{i,j} = \begin{cases} exp(-\alpha\|x_i - x_j\|^2) & if \quad \|x_i - x_j\| \leq \tau \\ 0 & if \quad otherwise \end{cases}$$

2. *Compute the* $d+1$ *bottom eigenvectors of the normalized Laplacian matrix* $\tilde{\Delta} = I - D^{-1/2} W D^{-1/2}$, $\tilde{\Delta} e_k = \lambda_k e_k$, $1 \leq k \leq d+1$, $0 = \lambda_1 \leq \cdots \leq \lambda_{d+1}$, *where* $D = diag(\sum_{k=1}^{n} W_{i,k})_{1 \leq i \leq n}$.

Dimension Reduction using Laplacian Eigenmaps

Algorithm - cont'd

Algorithm (Dimension Reduction using Laplacian Eigenmaps-cont'd)

3. *Construct the* $d \times n$ *matrix* Y*,*

$$Y = \begin{bmatrix} e_2^T \\ \vdots \\ e_{d+1}^T \end{bmatrix} D^{-1/2}$$

4. *The new geometric graph is obtained by converting the columns of* Y *into* n d*-dimensional vectors:*

$$\left[\begin{array}{c|c|c} y_1 & \cdots & y_n \end{array} \right] = Y$$

Output: $\{y_1, \cdots, y_n\} \subset \mathbb{R}^d$.

Example

see:
http://www.math.umd.edu/ rvbalan/TEACHING/AMSC663Fall2010/PROJECTS/P5/index.html

Dimension Reduction using LLE

The Idea

Presented in [1]. If data is sufficiently dense, we expect that each data point and its neighbors to lie on or near a (locally) linear patch. We assume we are given the set $\{x_1, \cdots, x_n\}$ in the high dimensional space $\mathbb{R}^N$.
Step 1. Find a set of local weights $B_{i,j}$ that best explain the point x_i from its local neighbors:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} \left\| x_i - \sum_j B_{i,j} x_j \right\|^2 \\ \text{subject to} & B_{i,j} \geq 0 \ \ , \ \ \sum_j B_{i,j} = 1 \ , \ i = 1, \cdots, n \end{array}$$

Step 2. Find the points $\{y_1, \cdots, y_n\} \subset \mathbb{R}^d$ that minimize

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} \left\| y_i - \sum_j B_{i,j} y_j \right\|^2 \\ \text{subject to} & \sum_{i=1}^{n} y_i = 0 \\ & \frac{1}{n} \sum_{i=1}^{n} y_i y_i^T = I_d \end{array}$$

Dimension Reduction using LLE

Derivations (1)

Step 1. The weights are obtained by solving a constrained least-squares problem. The optimization problem decouples for each i. The Lagrangian at fixed $i \in \{1, 2, \cdots, n\}$ is

$$L((B_{i,j})_{j \in N}, \lambda) = \|x_i - \sum_{j \in N} B_{i,j} x_j\|^2 + \lambda(\sum_{j \in N} B_{i,j} - 1) + \sum_{j \in N} \mu_j B_{i,j}$$

where N denotes its K-neighborhood of closest K vertices.
IF the nonegative constraint is not enforced, the Lagrangian simplifies to:

$$L() = \|\sum_{j \in N} B_{i,j}(x_i - x_j)\|^2 + \lambda(\sum_{j \in N} B_{i,j} - 1) = b^T C b + \lambda b^T \cdot 1 - \lambda$$

where C is the $K \times K$ covariance matrix $C_{j,k} = \langle x_j - x_i, x_k - x_i \rangle$ and $b = vect(B_{i,:})$. Set $\nabla_b L = 0$ and solve for b. $\nabla_b L = 2C \cdot b + \lambda 1$.

Dimension Reduction using LLE

Derivations (2)

$$b = -\frac{\lambda}{2} C^{-1} \cdot 1$$

The multiplier λ is obtained from the constraint $b^T \cdot 1 = 1$: $\lambda = -\frac{2}{1^T \cdot C^{-1} \cdot 1}$. Thus

$$b = \frac{C^{-1} \cdot 1}{1^T C^{-1} 1} \;,\;\; B = reshape(b)$$

IF the nonnegativity constraint is kept, then the problem becomes a Quadratic Optimization problem (use 'quadprog' in Matlab).

Step 2. The embedding in the lower dimensional space is obtained as follows. First denote $Y = [y_1 | \cdots | y_n]$ a $d \times n$ matrix. Then

$$\sum_{i=1}^{n} \| y_i - \sum_j B_{i,j} y_j \|^2 = \sum_{i=1}^{n} \langle y_i, y_i \rangle - 2 \sum_{i=1}^{n} \sum_j B_{i,j} \langle y_i, y_j \rangle + \sum_{i=1}^{n} \sum_{j,k} B_{i,j} B_{i,k} \langle y_j, y_k \rangle$$

$$= trace(YY^T) - 2trace(YBY^T) + trace(YB^T BY^T) =$$

Dimension Reduction using LLE

Derivations (3)

$$= trace(Y(I-B)^T(I-B)Y^T).$$

where B is the $n \times n$ (non-symmetric) matrix of weights. The optimization problem becomes:

$$\begin{array}{ll} \text{minimize} & trace(Y(I-B)^T(I-B)Y^T) \\ \text{subject to} & Y \cdot 1 = 0 \\ & YY^T = I_d \end{array}$$

Just as the graph Laplacian, the solution is given by the eigenvectors corresponding to the smallest eigenvalues of $(I-B)^T(I-B)$. The condition $Y \cdot 1 = 0$ rules out the lowest eigenvector (which is 1), and requires rows in Y to be orthogonal to this eigenvector. Therefore, the rows in Y are taken to be the eigenvectors associated to the smallest $d+1$ eigenvalues, except the smallest eigenvalue.

Dimension Reduction using LLE

Algorithm

Algorithm (Dimension Reduction using Locally Linear Embedding - Without non-negativity constraints)

Input: A geometric graph $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^N$. *Parameters: neighborhood size* K *and dimension* d.

1. *Finding the weight matrix* B*: For each point* i *do the following:*
   1. *Find its closest* K *neighbors, say* $\mathcal{V}_i$*; Let* $r : \mathcal{V}_i \to \{1, 2, \cdots, K\}$ *denote an indexing map;*
   2. *Compute the* $K \times K$ *local covariance matrix* C, $C_{j,k} = \langle x_j - x_i, x_k - x_i \rangle$.
   3. *Solve* $C \cdot u = 1$ *for* u *(*1 *denotes the* K*-vector of 1's).*
   4. *Rescale* $u = u/(u^T \cdot 1)$.
   5. *Set* $B_{i,r(j)} = u_j$ *for* $j \in \mathcal{V}_i$.

Dimension Reduction using LLE

Algorithm - cont'd

Algorithm (Dimension Reduction using Locally Linear Embedding)

2. *Solving the Eigen Problem:*
   1. *Create the (typically sparse) matrix* $L = (I - B)^T (I - B)$*;*
   2. *Find the bottom* $d + 1$ *eigenvectors of* L *(the bottom eigenvector whould be* $[1, \cdots, 1]^T$ *associated to eigenvalue 0)* $\{e_1, e_2, \cdots, e_{d+1}\}$*;*
   3. *Discard the last vector (the constant vector) and insert all other eigenvectors as rows into matrix* Y

$$Y = \begin{bmatrix} e_2^T \\ \vdots \\ e_{d+1}^T \end{bmatrix}$$

Output: $\{y_1, \cdots, y_n\} \subset \mathbb{R}^d$ *as columns from*

$$\left[\begin{array}{c|c|c} y_1 & \cdots & y_n \end{array} \right] = Y$$

Dimension Reduction using LLE

Algorithm

Algorithm (Dimension Reduction using Locally Linear Embedding - With non-negativity constraints)

Input: A geometric graph $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^N$. *Parameters: neighborhood size* K *and dimension* d.

1. *Finding the weight matrix* B*: For each point* i *do the following:*
   1. *Find its closest* K *neighbors, say* $\mathcal{V}_i$*; Let* $r : \mathcal{V}_i \to \{1, 2, \cdots, K\}$ *denote an indexing map;*
   2. *Compute the* $K \times K$ *local covariance matrix* C, $C_{r(j),r(k)} = \langle x_j - x_i, x_k - x_i \rangle$.
   3. *Solve for* u,
   $$\begin{array}{ll} \text{minimize} & u^T C u \\ \text{subject to} & u \geq 0 \ , \ u^T \cdot 1 = 1 \end{array}$$
   where 1 *denotes the* K*-vector of 1's.*
   4. *Set* $B_{i,j} = u_{r(j)}$ *for* $j \in \mathcal{V}_i$.

Dimension Reduction using LLE

Algorithm - cont'd

Algorithm (Dimension Reduction using Locally Linear Embedding)

2. *Solving the Eigen Problem:*
   1. *Create the (typically sparse) matrix* $L = (I - B)^T (I - B)$;
   2. *Find the bottom* $d + 1$ *eigenvectors of* L *(the bottom eigenvector whould be* $[1, \cdots, 1]^T$ *associated to eigenvalue 0)* $\{e_1, e_2, \cdots, e_{d+1}\}$;
   3. *Discard the last vector (the constant eigenvector) and insert all other eigenvectors as rows into matrix* Y

$$Y = \begin{bmatrix} e_2^T \\ \vdots \\ e_{d+1}^T \end{bmatrix}$$

Output: $\{y_1, \cdots, y_n\} \subset \mathbb{R}^d$ *as columns from*

$$\left[\begin{array}{c|c|c} y_1 & \cdots & y_n \end{array} \right] = Y$$

Dimension Reduction using Isomaps

The Idea

Presented in [2]. The idea is to first estimate all pairwise distances, and then use the nearly isometric embedding algorithm with full data that we will describe in the next lecture.
For each node in the graph we define the distance to the nearest K neighbors using the Euclidean metric. The distance to further nodes is defined as the geodesic distance w.r.t. these local distances.

Dimension Reduction using Isomaps

Algorithm

Algorithm (Dimension Reduction using Isomap)

Input: A geometric graph $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^N$. *Parameters: neighborhood size* K *and dimension* d.

1. *Construct the symmetric matrix* S *of squared pairwise distances:*
    1. *Construct the sparse matrix* T, *where for each node* i *find the nearest* K *neighbors* $\mathcal{V}_i$ *and set* $T_{i,j} = \|x_i - x_j\|_2$, $j \in \mathcal{V}_i$.
    2. *For any pair of two nodes* (i,j) *compute* $d_{i,j}$ *as the length of the shortest path,* $\sum_{p=1}^{L} T_{k_{p-1},k_p}$ *with* $k_0 = i$ *and* $k_L = j$, *using e.g. Dijkstra's algorithm.*
    3. *Set* $S_{i,j} = d_{i,j}^2$.

Dimension Reduction using Isomaps

Algorithm (Read this part of the algorithm after learning Isometric Embeddings)

Algorithm (Dimension Reduction using Isomap - cont'd)

2. *Compute the Gram matrix G:*

$$\rho = \frac{1}{2n} 1^T \cdot S \cdot 1 \ , \quad \nu = \frac{1}{n}(S \cdot 1 - \rho 1)$$

$$G = \frac{1}{2}\nu \cdot 1^T + \frac{1}{2} 1 \cdot \nu^T - \frac{1}{2} S$$

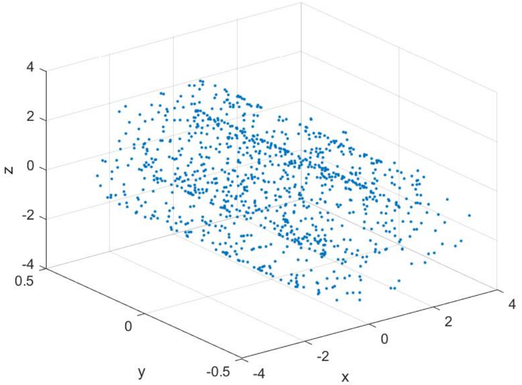
3. *Find the top d eigenvectors of G, say $e_1, \cdots, e_d$ so that $GE = E\Lambda$, form the matrix Y and then collect the columns:*

$$Y = \Lambda^{1/2} \begin{bmatrix} e_1^T \\ \vdots \\ e_d^T \end{bmatrix} = \begin{bmatrix} y_1 & | & \cdots & | & y_n \end{bmatrix}$$

Output: $\{y_1, \cdots, y_n\} \subset \mathbb{R}^d$.

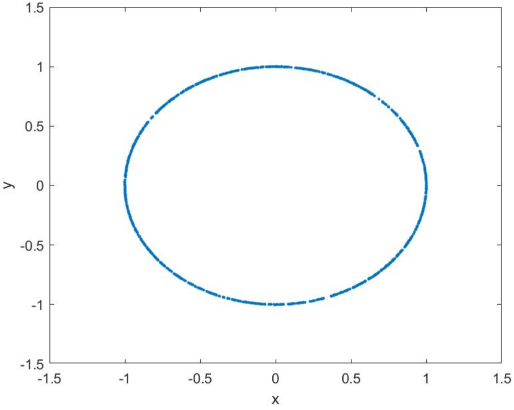
Data Sets

The Swiss Roll



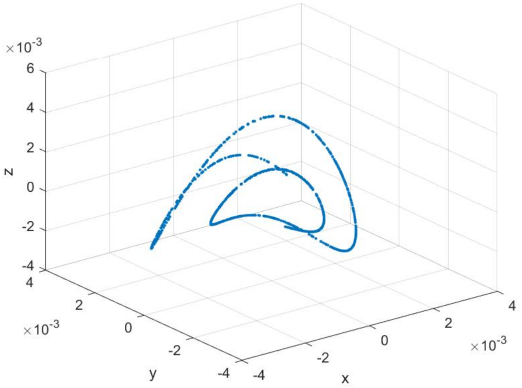
Data Sets

The Circle



Dimension Reduction for the Swiss Roll

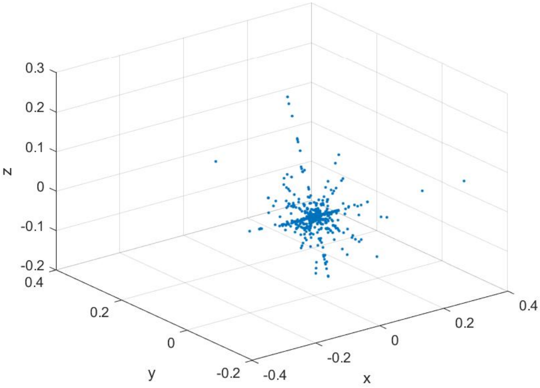
Laplacian Eigenmap



Parameters: $d = 3$, $W_{i,j} = exp(-0.1\|x_i - x_j\|^2)$, for all i, j.

Dimension Reduction for the Swiss Roll

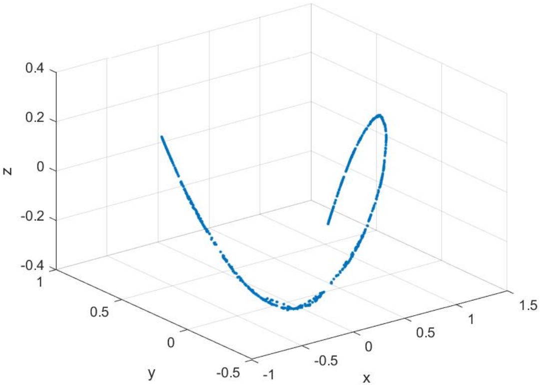
Local Linear Embedding (LLE)



Parameters: $d = 3$, $K = 2$.

Dimension Reduction for the Swiss Roll

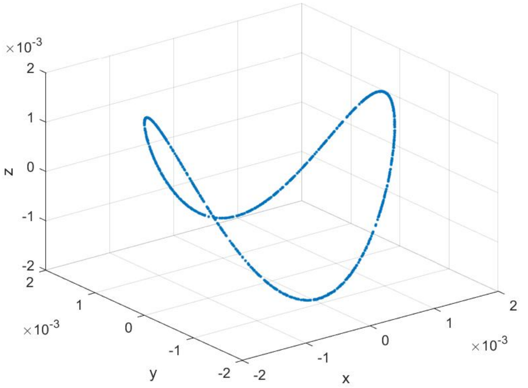
ISOMAP



Parameters: $d = 3$, $K = 10$.

Dimension Reduction for the Circle

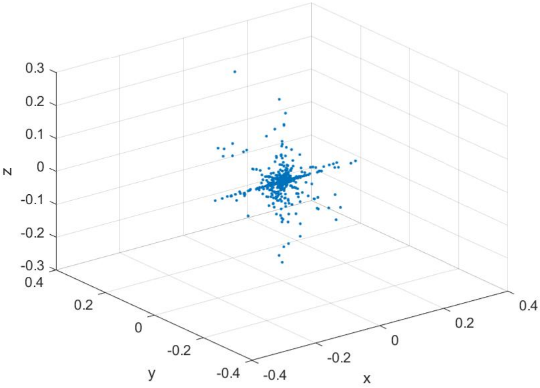
Laplacian Eigenmap



Parameters: $d = 3$, $W_{i,j} = exp(-0.1\|x_i - x_j\|^2)$, for all i, j.

Dimension Reduction for the Circle

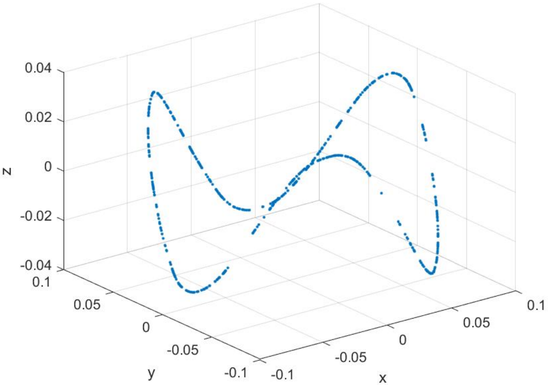
Local Linear Embedding (LLE)



Parameters: $d = 3$, $K = 2$.

Dimension Reduction for the Circle

ISOMAP



Parameters: $d = 3$, $K = 10$.

References

- S.T. Roweis, L.K. Saul, Locally linear embedding, Science 290, 2323–2326 (2000).
- J.B. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290, 2319–2323 (2000).