Lecture 10: Modern Multivariable Statistical Techniques & Manifold Learning

Math 178

Prof. Weiqing Gu

Today

- **Polynomial PCA**
- **Principal Curves and Surfaces**
- **ISOMAP**
- **LLE (Local Linear Embedding)**
- **Spectral Clustering**

Polynomial PCA

How should we generalize PCA to the nonlinear case? One possibility is to transform the set of input variables using a quadratic, cubic, or higher-degree polynomial, and then apply linear PCA (Gnanadesikan and Wilk, 1969). The resulting *polynomial PCA* again boils down to an eigenanalysis, but this time attention is focused on the *smallest* few eigenvalues for nonlinear dimensionality reduction.

In the quadratic PCA case, for example, the r-vector $\mathbf{X}$ is transformed into an extended r'-vector $\mathbf{X}'$, where $r' = 2r + r(r-1)/2$. Here, $\mathbf{X}'$ includes the original r variables plus r quadratic powers and $r(r-1)/2$ cross-products of the elements of $\mathbf{X}$. Thus, for the bivariate case ($r = 2$), quadratic PCA transforms $\mathbf{X} = (X_1, X_2)$ to $\mathbf{X}' = (X_1, X_2, X_1^2, X_2^2, X_1 X_2)$, and a linear PCA is carried out on the five transformed variables of $\mathbf{X}'$. If the bivariate observations follow an exact quadratic curve, the smallest eigenvalue of the covariance matrix of the extended vector will be zero, and the scores of the last principal component will be constant with a value of zero.

Example of Polynomial PCA

Consider, for example, the noiseless case in which $n = 201$ bivariate observations, (X_1, X_2), are generated to lie exactly on the quadratic curve $X_2 = 4X_1^2 + 4X_1 + 2$, where $X_1 = -1.5(0.01)0.5$. Suppose we carry out a linear PCA on the extended vector $(X_1^2, X_2^2, X_1, X_2, X_1X_2)$ and obtain five sets of principal component scores. See the upper panel of Table 16.1 for the eigenanalysis. The scatterplot matrix of the first four pairs of PC scores is given in Figure 16.1 and shows the pretzel-like shapes of the pairwise PCs. The last PC is not displayed because all its values are zero. The hyperplane defined by the zero eigenvalue is $0.696X_1 - 0.0174X_2 + 0.696X_1^2 = 0$ or $X_2 = 4X_1^2 + 4X_1$, which recovers the original quadratic curve (except for the constant). By varying the constant a, we can display a family of possible quadratic curves $X_2 = 4X_1^2 + 4X_1 + a$, and the constant a can be recovered from that curve that passes through each data point. The last PC (actually, $PC5/0.0174 + X_2$) is plotted in Figure 16.2 against X_1, for $a = 0, 1, 2, 3$, where we see that $a = 2$.

Suppose we now add standard Gaussian noise (mean 0, variance 1) independently to the X_1 and X_2-coordinates of each observation and then repeat the linear PCA on the resulting extended vector. How would the eigenanalysis and the PCA scatterplot matrix of the noiseless case be affected? For this noisy case, see the lower panel of Table 16.1. The eigenvalues are each greater than the respective eigenvalues from the noiseless case, with the smallest eigenvalue now 0.247. As we would expect, some of the well-defined patterns in the scatterplot matrix become blurred in the noisy case. Even if we significantly reduce the variance of the added noise component, the results of the quadratic PCA will still be strongly affected by the noisiness of the data.

TABLE 16.1. *Quadratic PCA for the bivariate data* (X_1, X_2), *where* $X_1 = -1.5(0.01)0.5$, $X_2 = 4X_1^2 + 4X_1 + 2$, *and* $n = 201$. *Eigenanalysis of the covariance matrix of the variables* $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$ *for the noiseless and noisy cases. The noisy case is obtained by replacing* X_1 *by* $X_1 + Z$ *and, independently,* X_2 *by* $X_2 + Z$, *where* $Z \sim \mathcal{N}(0, 1)$.

			Noiseless Case		
Eigenvalues	46.722	4.912	0.052	0.050	0.000
			Eigenvectors		
X_1	0.003	–0.253	0.620	0.115	0.696
X_2	–0.173	–0.013	0.337	–0.909	–0.174
X_1^2	–0.046	0.243	–0.578	–0.342	0.696
X_2^2	–0.979	–0.102	–0.063	0.165	0.000
X_1X_2	0.097	–0.929	–0.333	–0.129	0.000
			Noisy Case		
Eigenvalues	74.617	10.229	2.073	0.336	0.247
			Eigenvectors		
X_1	0.012	–0.271	–0.081	–0.380	–0.880
X_2	–0.165	0.000	0.009	–0.906	0.388
X_1^2	–0.019	0.357	–0.934	–0.014	–0.019
X_2^2	–0.980	–0.120	–0.027	0.160	–0.043
X_1X_2	0.121	–0.886	–0.348	0.089	0.268

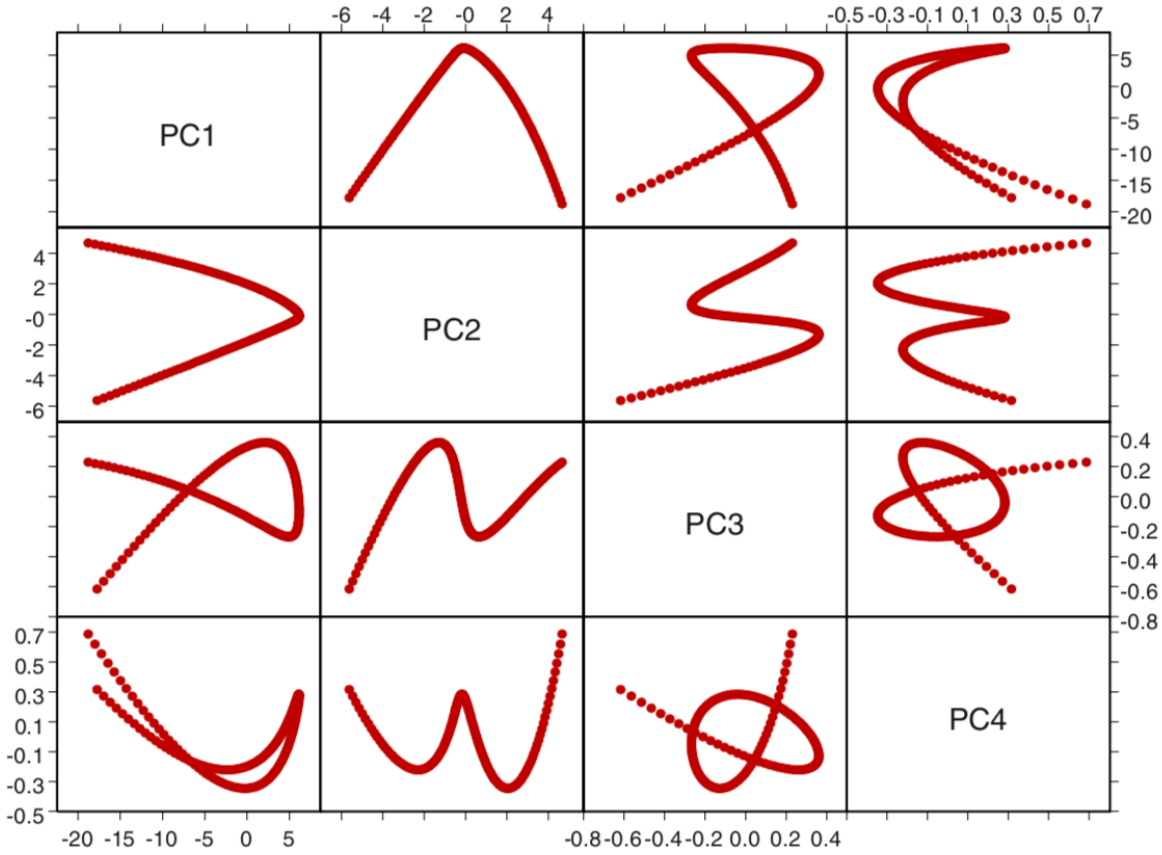


FIGURE 16.1. *Scatterplot matrix of the pairwise scores of the first four principal components from quadratic PCA using the covariance matrix. The last principal component has all its values equal to zero and is not displayed.*

MDS

- Work out details with the students on the board.

ISOMAP

- Original paper was published on Science
- https://web.mit.edu/cocosci/Papers/sci_reprint.pdf

The *isometric feature mapping* (or ISOMAP) algorithm (Tenenbaum, de Silva, and Langford, 2000) assumes that the smooth manifold $\mathcal{M}$ is a convex region of $\Re^t$ ($t \ll r$) and that the embedding $\psi : \mathcal{M} \to \mathcal{X}$ is an isometry. This assumption has two key ingredients:

- *Isometry:* The geodesic distance is invariant under the map ψ. For any pair of points on the manifold, $\mathbf{y}, \mathbf{y}' \in \mathcal{M}$, the geodesic distance between those points equals the Euclidean distance between their corresponding coordinates, $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$; i.e.,

$$d^{\mathcal{M}}(\mathbf{y}, \mathbf{y}') = \|\mathbf{x} - \mathbf{x}'\|_{\mathcal{X}}, \tag{16.36}$$

  where $\mathbf{y} = \phi(\mathbf{x})$ and $\mathbf{y}' = \phi(\mathbf{x}')$.

- *Convexity:* The manifold $\mathcal{M}$ is a convex subset of $\Re^t$.

Thus, ISOMAP regards $\mathcal{M}$ as a convex region that may have been distorted in any of a number of ways (e.g., by folding or twisting). The so-called *Swiss roll*,[3] which is a flat two-dimensional rectangular submanifold of $\Re^3$, is one such example; see Figure 16.7. ISOMAP appears to work best for intrinsically flat submanifolds of $\mathcal{X} = \Re^r$ that look like rolled-up sheets of paper. In certain situations, the isometry assumption appears to be reasonable, while the convexity assumption may be too restrictive (Donoho and Grimes, 2003).

ISOMAP uses the isometry and convexity assumptions to form a nonlinear generalization of multidimensional scaling (MDS). As we saw in Section 13.3, MDS searches for a low-dimensional subspace in which to embed input data while preserving the Euclidean interpoint distances. ISOMAP extends the MDS paradigm by attempting to preserve the global geometry properties of the underlying nonlinear manifold, and it does this by approximating *all* geodesic distances (i.e., lengths of the shortest paths) on the manifold. In this sense, ISOMAP gives a *global* approach to manifold learning.

The Isomap algorithm consists of three steps:

1. Neighborhood graph. Fix either an integer K or an $\epsilon > 0$. Calculate the distances,

$$d_{ij}^{\mathcal{X}} = d^{\mathcal{X}}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_{\mathcal{X}}, \tag{16.37}$$

between all pairs of data points $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}$, $i, j = 1, 2, \ldots, n$. These are generally taken to be Euclidean distances but may be a different distance metric. Determine which data points are "neighbors" on the manifold $\mathcal{M}$ by connecting each point either to its K nearest neighbors or to all points lying within a ball of radius ϵ of that point. Choice of K or ϵ controls neighborhood size and also the success of Isomap.

This gives us a *weighted neighborhood graph* $\mathcal{G} = \mathcal{G}(\mathcal{V}, \mathcal{E})$, where the set of *vertices* $\mathcal{V} = \{\mathbf{x}_1 \ldots, \mathbf{x}_n\}$ are the input data points, and the set of *edges* $\mathcal{E} = \{e_{ij}\}$ indicate neighborhood relationships between the points. The edge e_{ij} that joins the neighboring points $\mathbf{x}_i$ and $\mathbf{x}_j$ has a weight w_{ij} associated with it, and that weight is given by the "distance" $d_{ij}^{\mathcal{X}}$ between those points. If there is no edge present between a pair of points, the corresponding weight is zero.

2. Compute graph distances. Estimate the unknown true *geodesic distances*, $\{d_{ij}^{\mathcal{M}}\}$, between pairs of points in $\mathcal{M}$ by *graph distances*, $\{d_{ij}^{\mathcal{G}}\}$, with respect to the graph $\mathcal{G}$. The graph distances are the shortest path distances between all pairs of points in the graph $\mathcal{G}$. Points that are not neighbors of each other are connected by a sequence of neighbor-to-neighbor links, and the length of this path (sum of the link weights) is taken to approximate the distance between its endpoints on the manifold.

If the data points are sampled from a probability distribution that is supported by the entire manifold, then, asymptotically (as $n \to \infty$), it turns out that the estimate $d^{\mathcal{G}}$ converges to $d^{\mathcal{M}}$ if the manifold is flat (Bernstein, de Silva, Langford, and Tenenbaum, 2001).

An efficient algorithm for computing the shortest path between every pair of vertices in a graph is *Floyd's algorithm* (Floyd, 1962), which works best with dense graphs (graphs with many edges).

3. Embedding via multidimensional scaling. Let $\mathbf{D}^{\mathcal{G}} = (d_{ij}^{\mathcal{G}})$ be the symmetric $(n \times n)$-matrix of graph distances. Apply "classical" MDS to $\mathbf{D}^{\mathcal{G}}$ to give the reconstructed data points in a t-dimensional feature space $\mathcal{Y}$, so that the geodesic distances on $\mathcal{M}$ between data points is preserved as much as possible:

- Form the "doubly centered," symmetric, $(n \times n)$-matrix of squared graph distances,

$$\mathbf{A}_n^{\mathcal{G}} = -\frac{1}{2}\mathbf{H}\mathbf{S}^{\mathcal{G}}\mathbf{H}, \tag{16.38}$$

  where $\mathbf{S}^{\mathcal{G}} = ([d_{ij}^{\mathcal{G}}]^2)$, $\mathbf{H} = \mathbf{I}_n - n^{-1}\mathbf{J}_n$, and $\mathbf{J}_n = \mathbf{1}_n\mathbf{1}_n^{\tau}$ is an $(n \times n)$-matrix of ones. The matrix $\mathbf{A}_n^{\mathcal{G}}$ will be nonnegative-definite of rank $t < n$.

- The embedding vectors $\{\widehat{\mathbf{y}}_i\}$ are chosen to minimize $\|\mathbf{A}_n^{\mathcal{G}} - \mathbf{A}_n^{\mathcal{Y}}\|$, where $\mathbf{A}_n^{\mathcal{Y}}$ is (16.38) with $\mathbf{S}^{\mathcal{Y}} = ([d_{ij}^{\mathcal{Y}}]^2)$ replacing $\mathbf{S}^{\mathcal{G}}$, and $d_{ij}^{\mathcal{Y}} = \|\mathbf{y}_i - \mathbf{y}_j\|$ is the Euclidean distance between $\mathbf{y}_i$ and $\mathbf{y}_j$. The optimal solution is given by the eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_t$ corresponding to the t largest (positive) eigenvalues, $\lambda_1 \geq \cdots \geq \lambda_t$, of $\mathbf{A}_n^{\mathcal{G}}$.

- The graph $\mathcal{G}$ is embedded into $\mathcal{Y}$ by the $(t \times n)$-matrix

$$\widehat{\mathbf{Y}} = (\widehat{\mathbf{y}}_1, \cdots, \widehat{\mathbf{y}}_n) = (\sqrt{\lambda_1}\mathbf{v}_1, \cdots, \sqrt{\lambda_t}\mathbf{v}_t)^{\tau}. \tag{16.39}$$

  The ith column of $\widehat{\mathbf{Y}}$ yields the embedding coordinates in $\mathcal{Y}$ of the ith data point. The Euclidean distances between the n t-dimensional columns of $\widehat{\mathbf{Y}}$ are collected into the $(n \times n)$-matrix $\mathbf{D}_t^{\mathcal{Y}}$.

The ISOMAP algorithm appears to work most efficiently with $n \leq 1{,}000$. Changes to the ISOMAP code (see below) enable us to work with much larger data sets.

As a measure of how closely the ISOMAP t-dimensional solution matrix $\mathbf{D}_t^{\mathcal{Y}}$ approximates the matrix $\mathbf{D}^{\mathcal{G}}$ of graph distances, we plot $1 - R_t^2$ against dimensionality t (i.e., $t = 1, 2, \ldots, t^*$, where t^* is some integer such as 10), where $R_t^2 = [\text{corr}(\mathbf{D}_t^{\mathcal{Y}}, \mathbf{D}^{\mathcal{G}})]^2$ is the squared correlation coefficient of all corresponding pairs of entries in the matrices $\mathbf{D}_t^{\mathcal{Y}}$ and $\mathbf{D}^{\mathcal{G}}$. The intrinsic dimensionality is taken to be that integer t at which an "elbow" appears in the plot.

Consider, for example, the two-dimensional Swiss roll manifold embedded in three-dimensional space. Suppose we are given 20,000 points randomly drawn from the surface of that manifold.[4] The 3D scatterplot of the data is given in the right panel of Figure 16.7. Using all 20,000 points as input to the ISOMAP algorithm proves to be computationally too big, and so we use only the first 1000 points for illustration. Taking $n = 1,000$ and $K = 7$ neighborhood points, Figure 16.8 shows a plot of the values of $1-R_t^2$ against t for $t = 1, 2, \ldots, 10$, where an elbow correctly shows $t = 2$; the 2D ISOMAP neighborhood-graph solution is given in Figure 16.9.

The ISOMAP algorithm has difficulty with manifolds that contain holes, have too much curvature, or are not convex. In the case of noisy data, it depends upon how the neighborhood size (either K or ϵ) is chosen; if K or ϵ are chosen neither too large (that it introduces false connections into $\mathcal{G}$) nor too small (that $\mathcal{G}$ becomes too sparse to approximate geodesic paths accurately), then ISOMAP should be able to tolerate moderate amounts of noise in the data.

Landmark Isomap

When a data set is very large, the performance of the ISOMAP algorithm is significantly degraded by having to store in memory the complete $(n \times n)$-matrix $\mathbf{D}_G$ (step 2) and carry out an eigenanalysis of the $(n \times n)$-matrix $\mathbf{A}_n$ for the MDS reconstruction (step 3). If the data are truly scattered around a low-dimensional manifold, then the vast majority of pairwise distances will be redundant; to speed up the MDS embedding step, we have to eliminate as many of the redundant distance calculations as we can.

In LANDMARK ISOMAP, the researcher tries to eliminate such redundancy by specifying a *landmark* subset of m of the n data points (de Silva and Tenenbaum, 2003). For example, if $\mathbf{x}_i$ is designated as one of the m landmark points, we calculate only those distances between each of the n points and $\mathbf{x}_i$. Input to the LANDMARK ISOMAP algorithm is, therefore, an $(m \times n)$-matrix of distances. The landmark points may be selected by random sampling or by a judicious choice of "representative" points. The number of such landmark points is left to the researcher, but $m = 50$ works well. In the MDS embedding step, the object is to preserve only those d tances between all points and the subset of landmark points. Step 2 LANDMARK ISOMAP uses *Dijkstra's algorithm* (Dijkstra, 1959), which faster than Floyd's algorithm for computing graph distances and is g erally preferred when the graph is sparse. Dijkstra's algorithm is also r ommended as a replacement for Floyd's algorithm in the original ISOM algorithm.

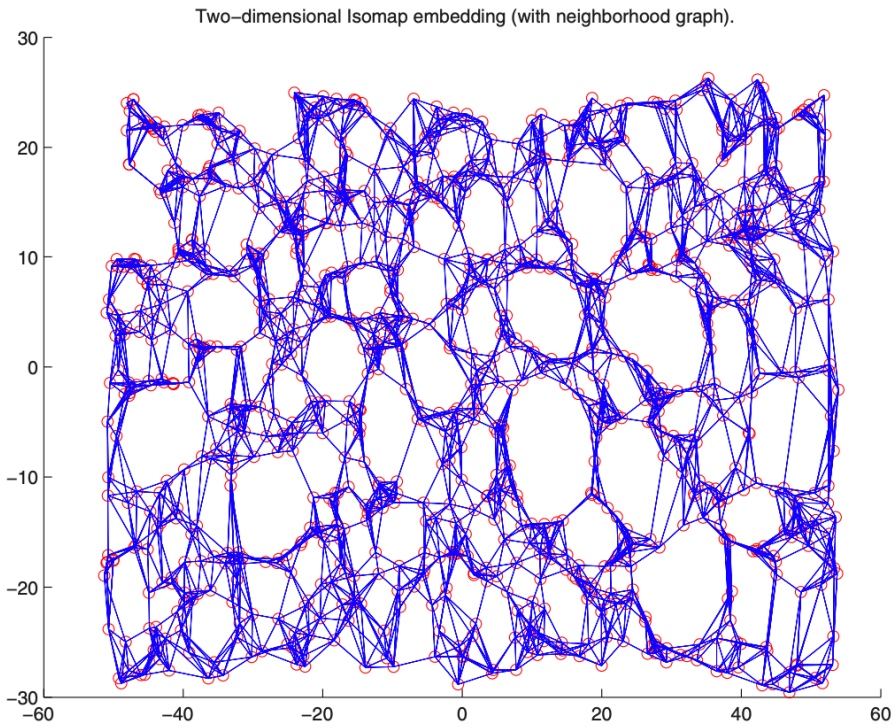


FIGURE 16.9. *Two-dimensional* ISOMAP *embedding, with neighborhood graph, of the* $n = 1{,}000$ *Swiss roll data points. The number of neighborhood points is* $K = 7$.

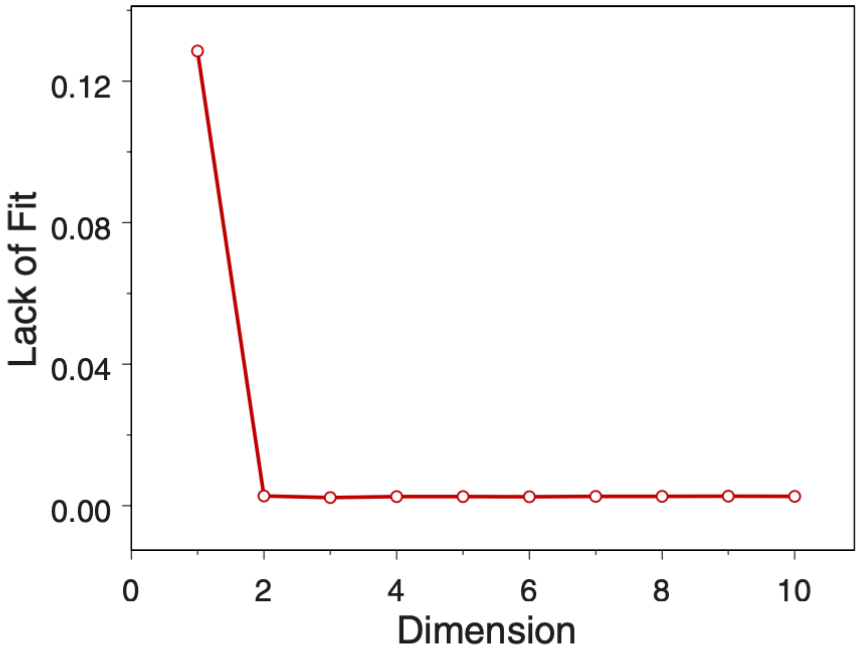
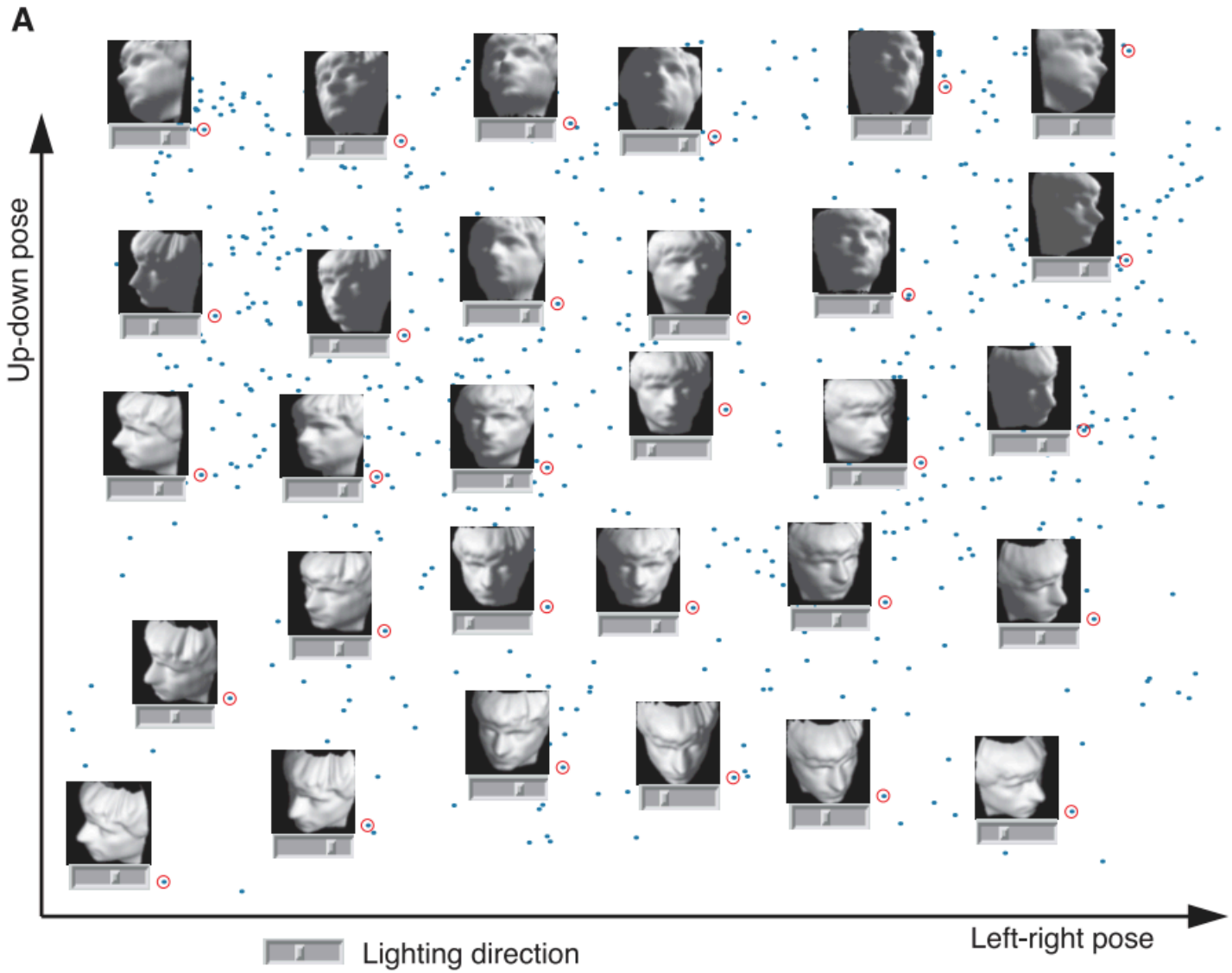


FIGURE 16.8. ISOMAP *dimensionality plot for the* $n = 1{,}000$ *Swiss roll data points. The number of neighborhood points is* $K = 7$*. The plotted points are* $(t, 1 - R_t^2)$*,* $t = 1, 2, \ldots, 10$*.*

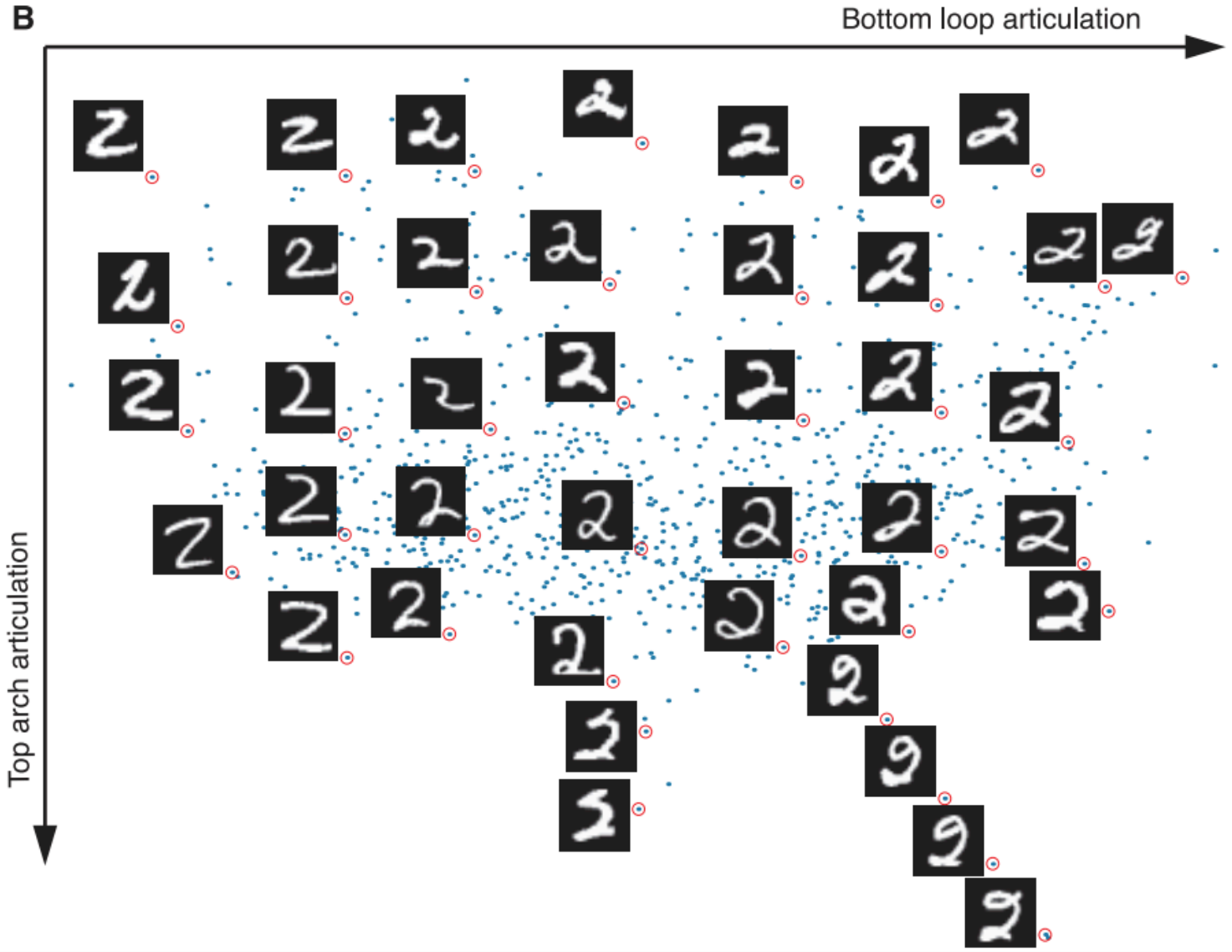
Applying LANDMARK ISOMAP to the $n = 1{,}000$ Swiss roll data points with $K = 7$ and the first $m = 50$ points taken to be landmark points results in an elbow at $t = 2$ in the dimensionality plot; the 2D LANDMARK ISOMAP neighborhood-graph solution is given in Figure 16.10. This is a much faster solution than the one we obtained using the original ISOMAP algorithm. The main differences between Figures 16.9 and 16.10 are roundoff error and a rotation due to sign changes.

Because of the significant increase in computational speed, we can apply LANDMARK ISOMAP to all 20,000 points (using $K = 7$ and $m = 50$); an elbow again correctly appears at $t = 2$ in the dimensionality plot, and the resulting 2D LANDMARK ISOMAP neighborhood-graph solution is given in Figure 16.11.

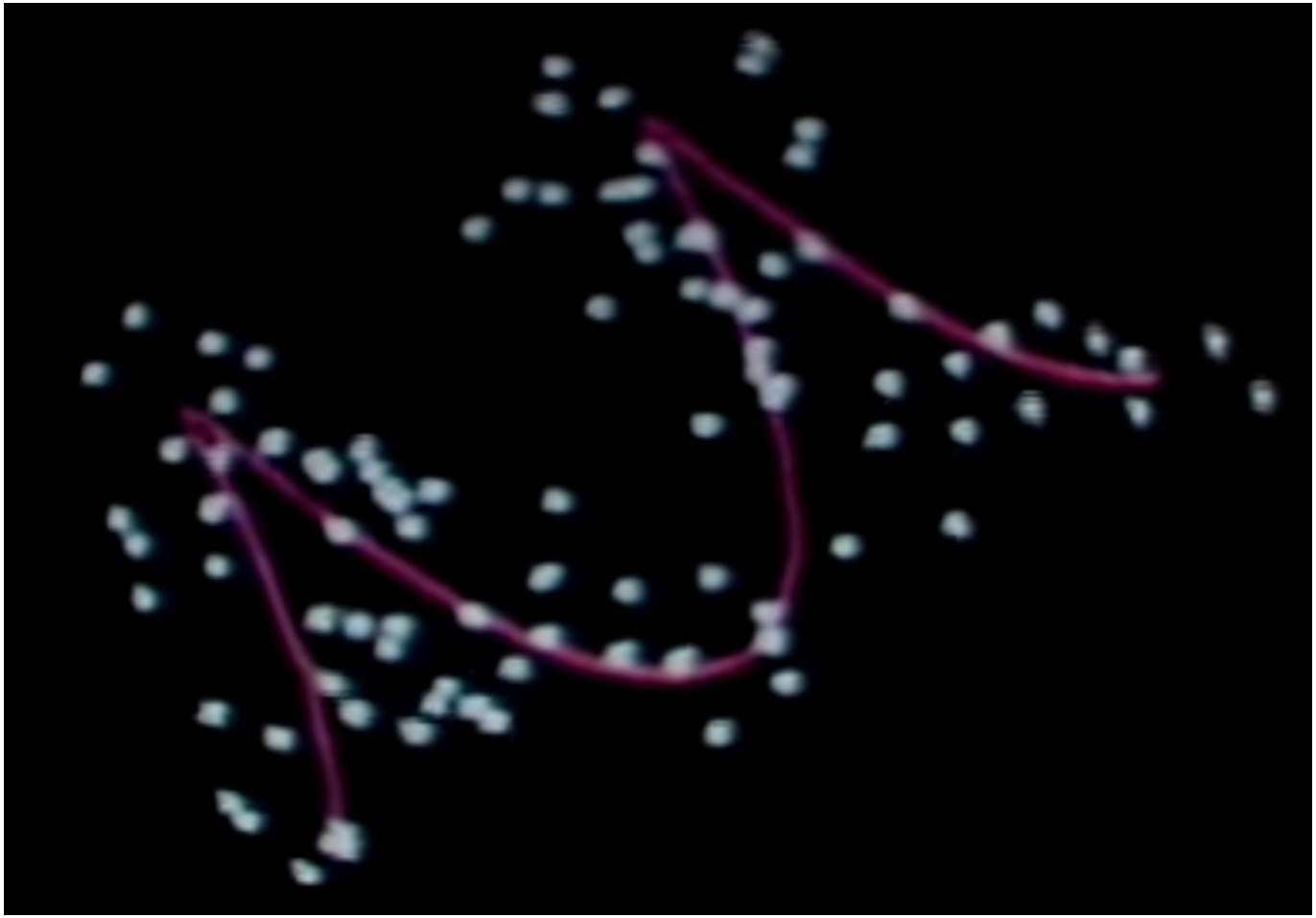
Results of ISOMAP



Result of ISOMAP



Next, how to fit a data by a 3D curve?



Projection Index

Consider a data point $\mathbf{x} \in \Re^r$ and let $\mathbf{f}(\lambda)$ be a curve. Project $\mathbf{x}$ to a point on $\mathbf{f}(\lambda)$ that is closest (in Euclidean distance) to $\mathbf{x}$. Let

$$\lambda_{\mathbf{f}}(\mathbf{x}) = \sup_{\lambda} \left\{ \lambda : \|\mathbf{x} - \mathbf{f}(\lambda)\| = \inf_{\mu} \|\mathbf{x} - \mathbf{f}(\mu)\| \right\} \tag{16.5}$$

be the *projection index*, $\lambda_{\mathbf{f}} : \Re^r \to \Re$, which produces a value of λ for which $\mathbf{f}(\lambda)$ is closest to $\mathbf{x}$. In the unlikely event that there are multiple points on the curve closest to $\mathbf{x}$ (called *ambiguity points*), the projection index will pick that point with the largest value of the projection index. Note that $\lambda_{\mathbf{f}}$ can be a discontinuous function.

Note: A curve in R^r is just an extension of a curve in R^3.

Curve in R^3 => Curve in R^r

A one-dimensional curve in an r-dimensional space is an analogue of a straight line in $\Re^r$. To formalize this notion, we define a one-dimensional curve in $\Re^r$ as a function $\mathbf{f} : \Lambda \to \Re^r$, for $\Lambda \subseteq \Re$, so that

$$\mathbf{f}(\lambda) = (f_1(\lambda), \cdots, f_r(\lambda))^\tau \tag{16.1}$$

is an r-vector parameterized by $\lambda \in \Lambda$. For example, the unit circle in $\Re^2$, $\{(x_1, x_2) \in \Re^2 : x_1^2 + x_2^2 = 1\}$, is a one-dimensional curve that can be parameterized as

$$\mathbf{f}(\lambda) = (f_1(\lambda), f_2(\lambda))^\tau = (\cos\lambda, \sin\lambda)^\tau, \quad \lambda \in [0.2\pi). \tag{16.2}$$

Principal Curves

We define the *reconstruction error* as the expected squared distance between $\mathbf{X}$ (or its associated density) and $\mathbf{f}$,

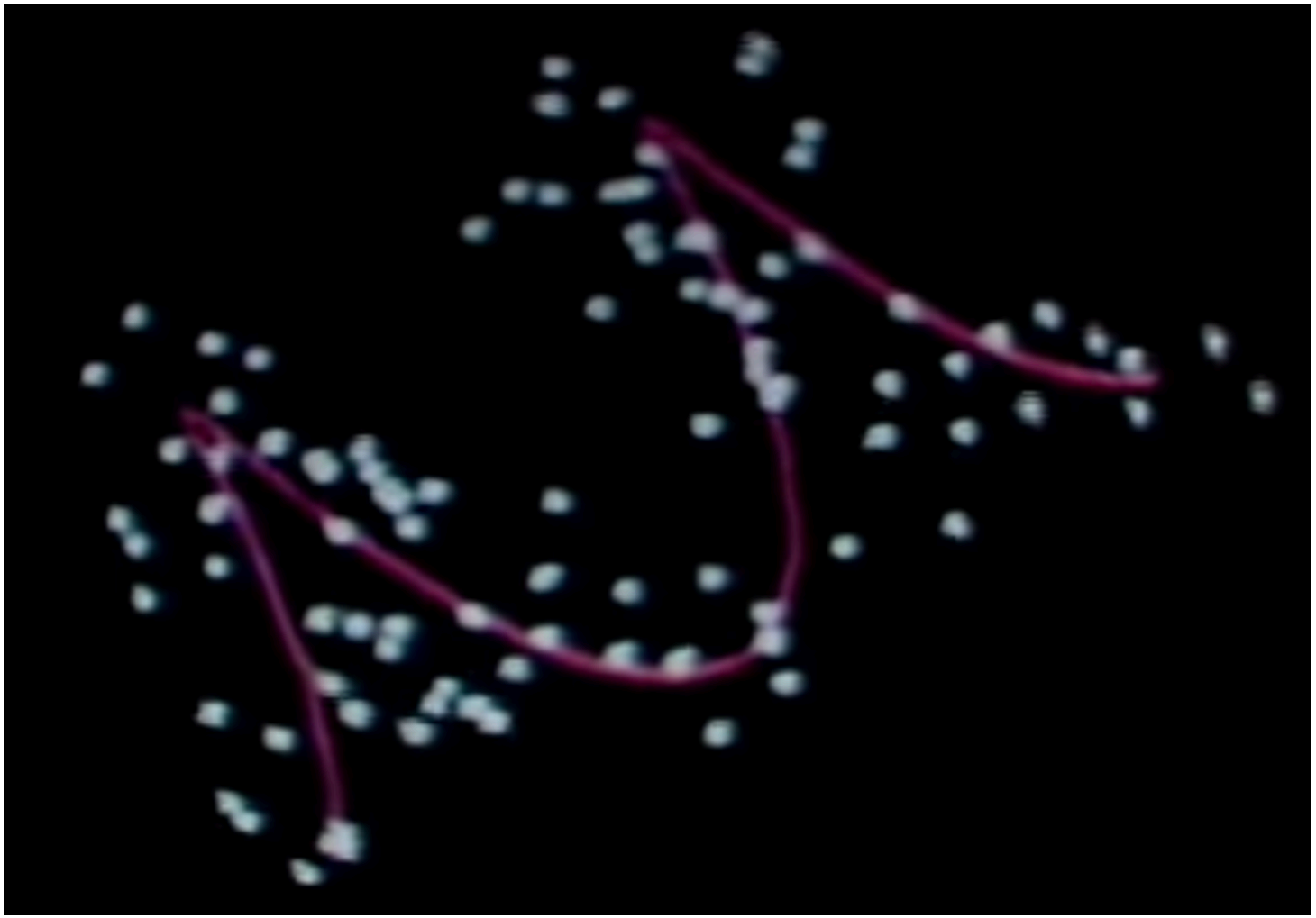
$$D^2(\mathbf{X}, \mathbf{f}) = \mathrm{E}\left\{\|\mathbf{X} - \mathbf{f}(\lambda_{\mathbf{f}}(\mathbf{X}))\|^2\right\}. \tag{16.6}$$

If $\mathbf{f}(\lambda)$ satisfies

$$\mathbf{f}(\lambda) = \mathrm{E}\{\mathbf{X}|\lambda_{\mathbf{f}}(\mathbf{X}) = \lambda\}, \quad \text{for almost every } \lambda \in \Lambda, \tag{16.7}$$

then $\mathbf{f}(\lambda)$ is said to be *self-consistent* or a principal curve for $\mathbf{X}$ (or its associated density $p_{\mathbf{X}}$). Thus, for any point on the curve, $\mathbf{f}(\lambda)$ is the average of all those data values that project to that point.

Example: Principal Curve in R^3

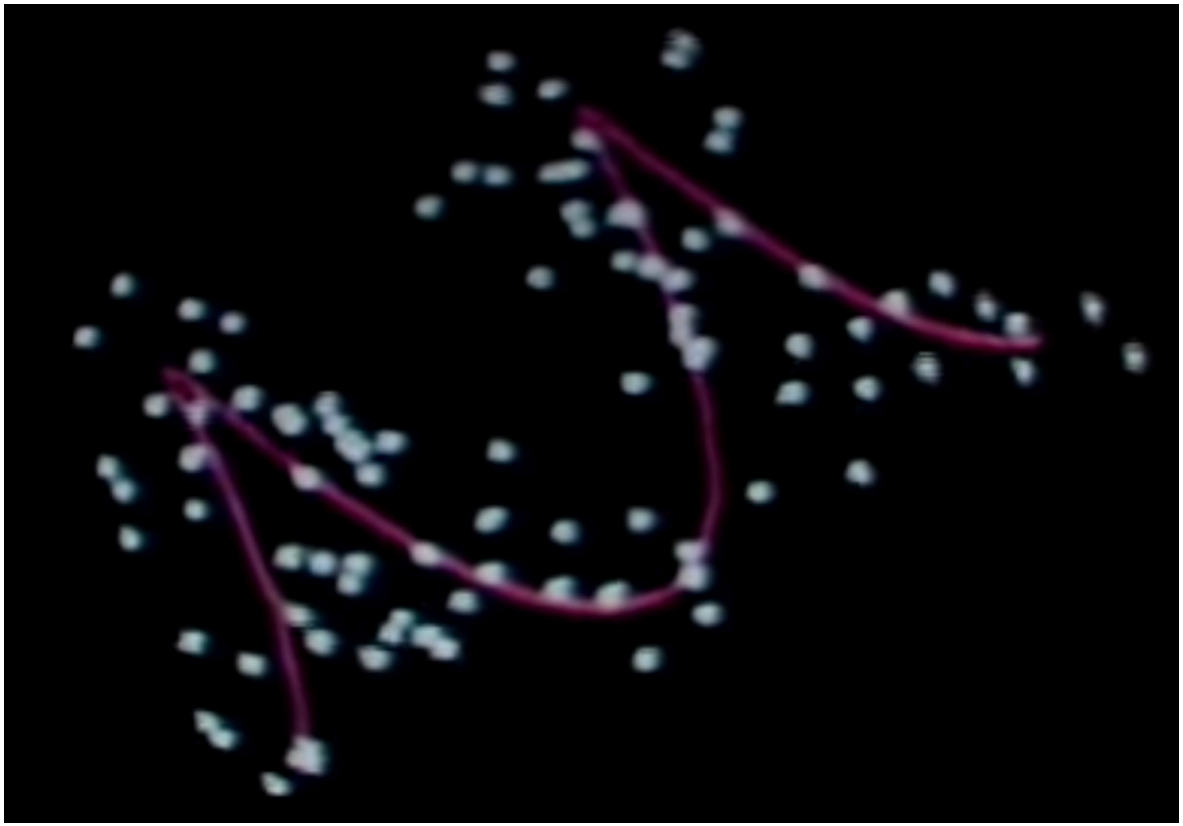


https://www.youtube.com/watch?v=xhHe0C2iUsY

How to find Principal Curve?

Projection-Expectation Algorithm

Basically an Expectation and Maximization Algorithm!



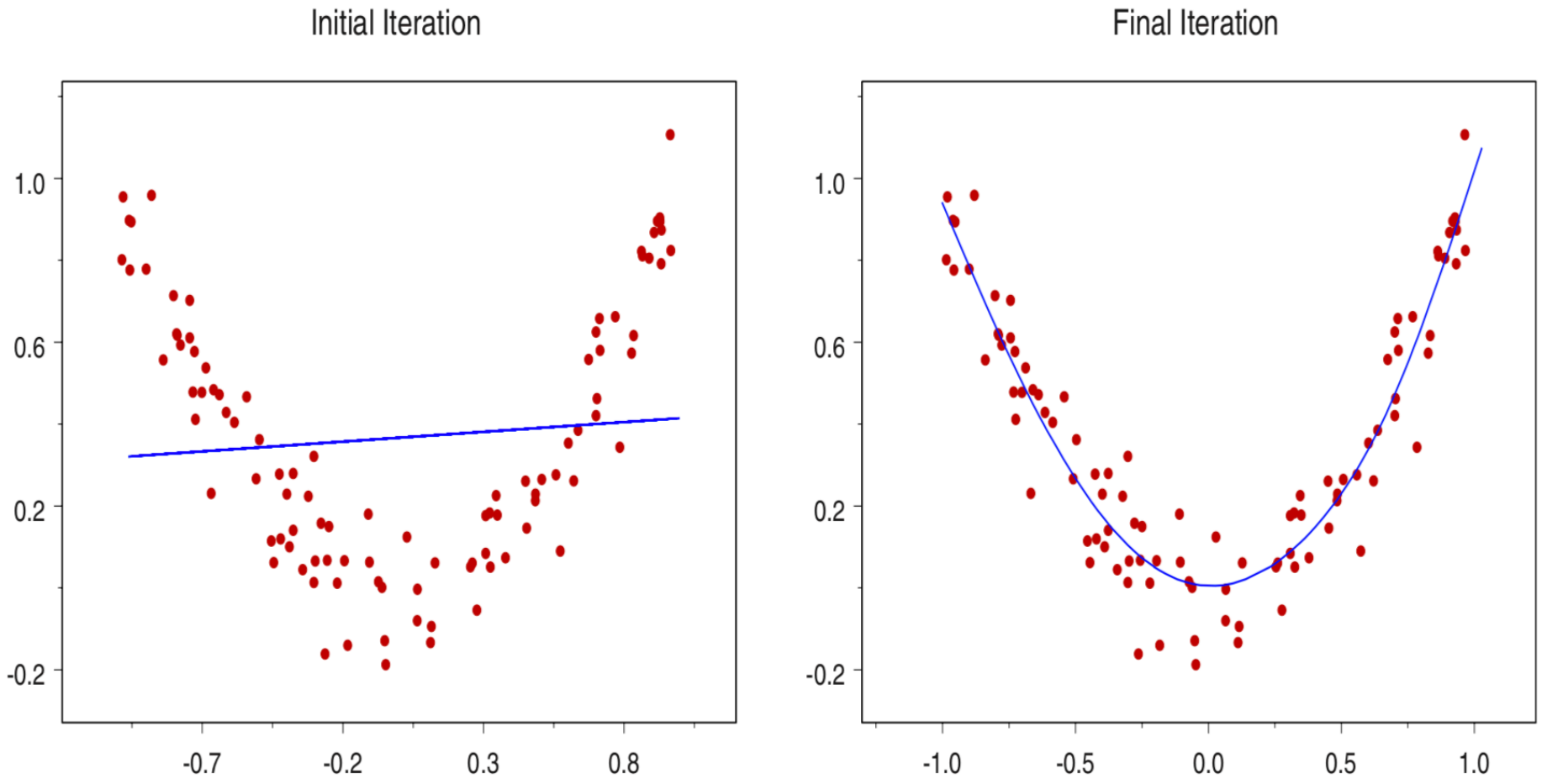


FIGURE 16.3. *Principal curve fitted to 100 randomly generated observations in two dimensions, where X_2 is a quadratic function of X_1 plus Gaussian noise with mean 0 and standard deviation 0.1. Left panel: initial iteration, first principal component, $D^2 = 1023.3$. Right panel: final iteration, principal curve, $D^2 = 0.54$.*

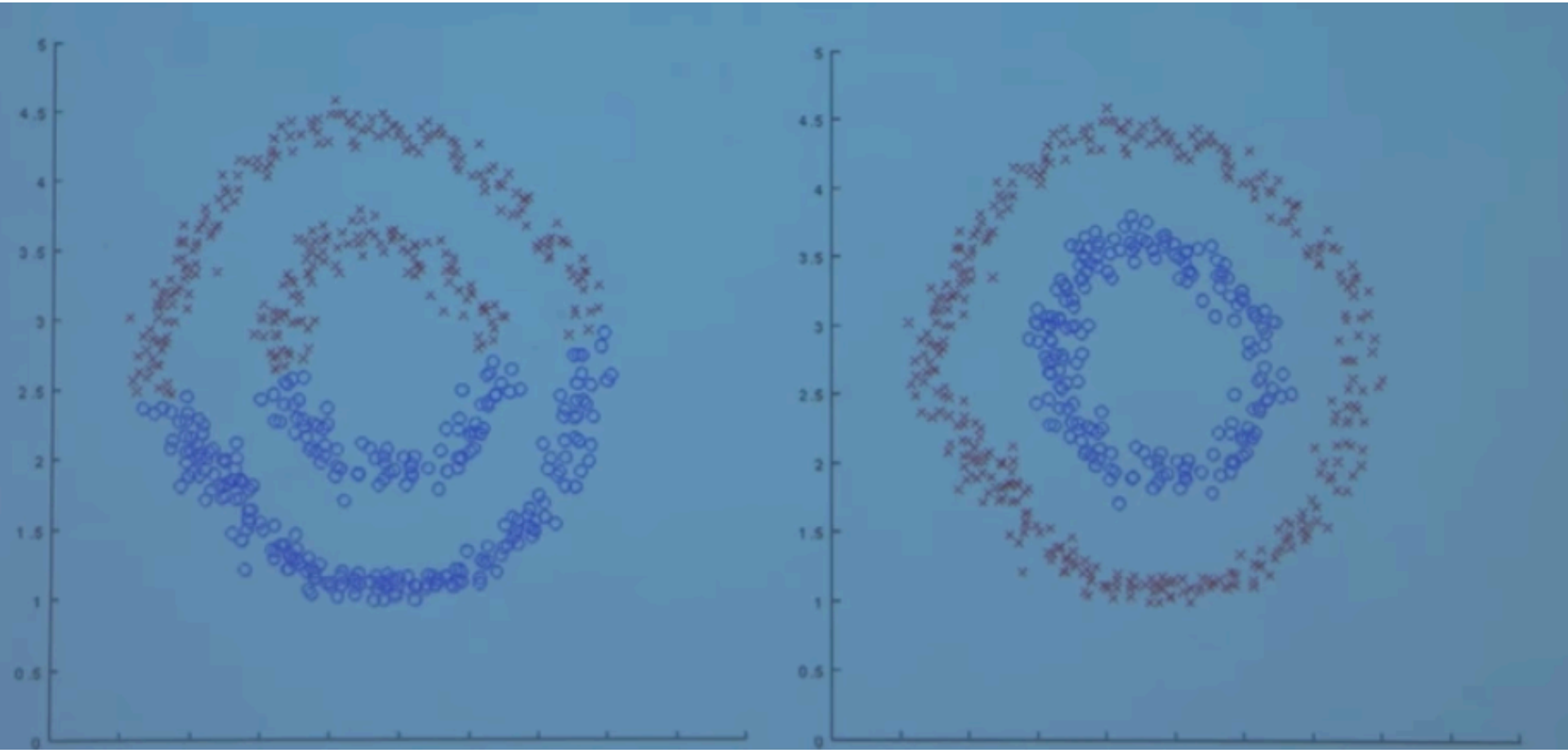
Principal Surfaces

- See the video.
- https://www.youtube.com/watch?v=xhHe0C2iUsY

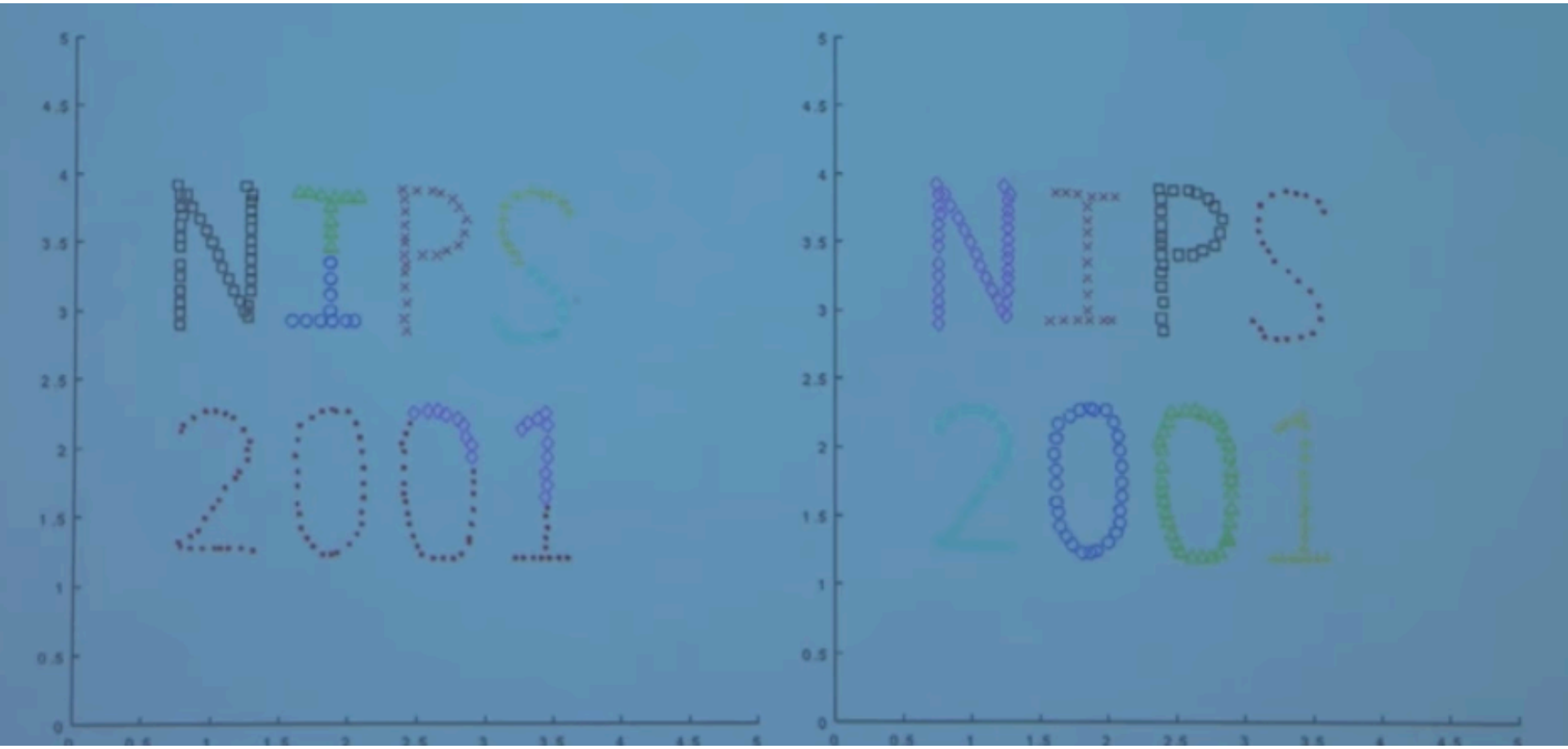
Spectral Clustering

- Work out details with students on the board.

Result of spectral clustering on the right K-mean on the left



Result of spectral clustering on the right K-mean on the left



Laplacian Eigenmaps:

`people.cs.uchicago.edu/~misha/ManifoldLearning/index.html`

HLLE: `basis.stanford.edu/WWW/HLLE/frontdov.htm`

See Martinez and Martinez (2005, Section 3.2 and Appendix B). There is also a `Matlab_Toolbox_for_Dimensionality_Reduction`, which is downloadable from the website

`www.cs.unimaas.nl/l.vandermaaten/Laurens_van_der_Maaten`

and includes all the methods discussed in this chapter and many data sets. There is, at present, no S-PLUS/R code for ISOMAP, LLE, Laplacian eigenmaps, or HLLE.

Software

The website `www.iro.umontreal.ca/~kegl/research/pcurves` gives a review of the area of principal curves and gives an introduction to algorithms and software. The S-PLUS/R computer packages `princurve` and `pcurve`, both based on *S*-code originally written by Hastie, are available for fitting a principal curve to multivariate data. MATLAB code for principal curves is available at `lear.inrialpes.fr/ verbeek/software`.

There are several publicly available computer programs for performing kernel PCA; see, for example, the `kcpa` function included in the R package `kernlab`, which can be downloaded from CRAN.

MATLAB code for implementing ISOMAP, LLE, and HLLE is publicly available at the following websites:

ISOMAP: `isomap.stanford.edu`

LLE: `www.cs.toronto.edu/~roweis/lle/`

- Please check this out on manifold learning with codes.

https://scikit-learn.org/stable/modules/manifold.html

Previous 2.1. Gaussian... | Next 2.3. Clustering | Up 2. Unsupervis...

scikit-learn v0.21.2
Other versions

Please **cite us** if you use the software.

2.2. Manifold learning
2.2.1. Introduction
2.2.2. Isomap
- 2.2.2.1. Complexity

2.2.3. Locally Linear Embedding
- 2.2.3.1. Complexity

2.2.4. Modified Locally Linear Embedding
- 2.2.4.1. Complexity

2.2.5. Hessian Eigenmapping
- 2.2.5.1. Complexity

2.2.6. Spectral Embedding
- 2.2.6.1. Complexity

2.2.7. Local Tangent Space Alignment
- 2.2.7.1. Complexity

2.2.8. Multi-dimensional Scaling (MDS)
- 2.2.8.1. Metric MDS
- 2.2.8.2. Nonmetric MDS

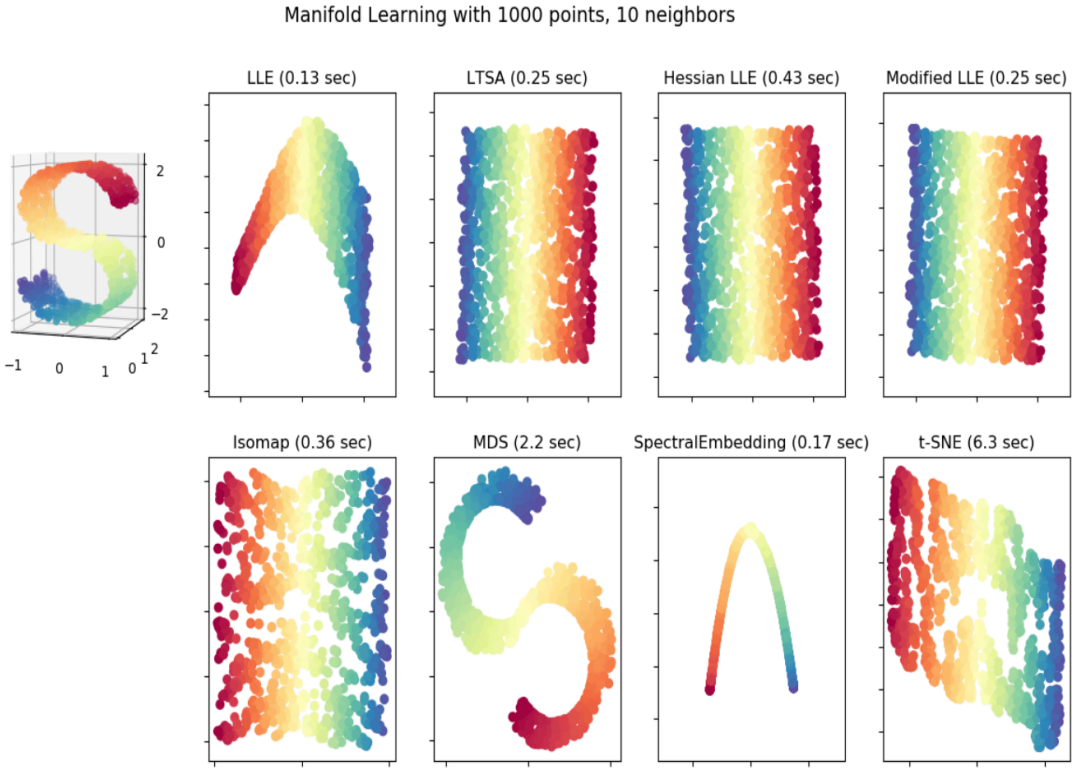
2.2.9. t-distributed Stochastic Neighbor Embedding (t-SNE)
- 2.2.9.1. Optimizing t-SNE
- 2.2.9.2. Barnes-Hut t-SNE

2.2.10. Tips on practical use

2.2. Manifold learning

Look for the bare necessities
The simple bare necessities
Forget about your worries and your strife
I mean the bare necessities
Old Mother Nature's recipes
That bring the bare necessities of life

– Baloo's song [The Jungle Book]



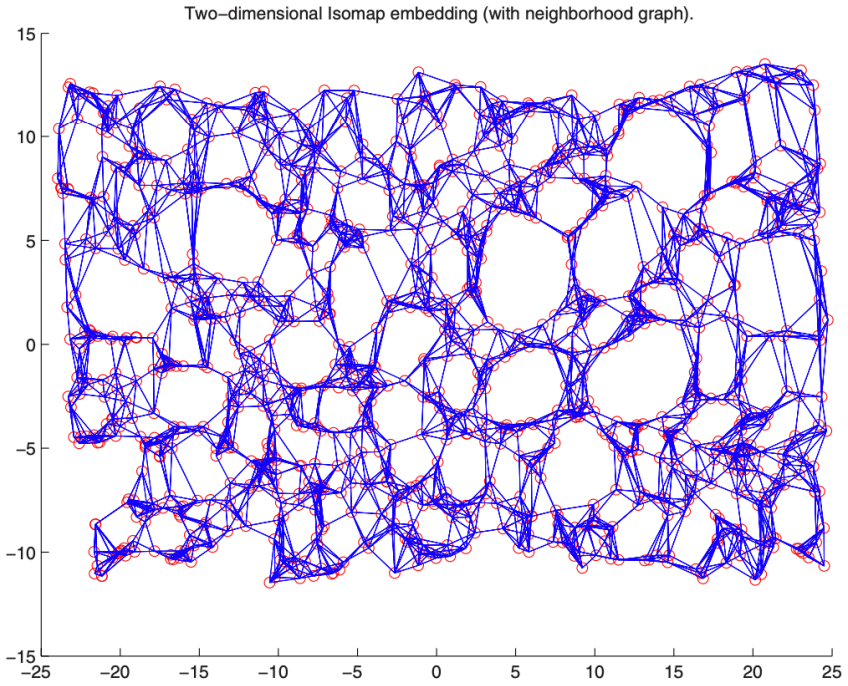


FIGURE 16.10. *Two-dimensional* LANDMARK ISOMAP *embedding, with neighborhood graph, of the* $n = 1{,}000$ *Swiss roll data points. The number of neighborhood points is* $K = 7$ *and the number of landmark points is* $m = 50$.

Local Linear Embedding

The *local linear embedding (LLE)* algorithm (Roweis and Saul, 2000; Saul and Roweis, 2003) for nonlinear dimensionality reduction is similar in spirit to the ISOMAP algorithm, but because it attempts to preserve local neighborhood information on the (Riemannian) manifold (without estimating the true geodesic distances), we view LLE as a *local* approach rather than as the ISOMAP's global approach.

Like ISOMAP, the LLE algorithm also consists of three steps:

1. Nearest neighbor search. Fix $K \ll r$ and let N_i^K denote the "neighborhood" of $\mathbf{x}_i$ that contains only its K nearest points, as measured by Euclidean distance (K could be different for each point $\mathbf{x}_i$).

The success of LLE depends (as does ISOMAP) upon the choice of K: it must be sufficiently large so that the points can be well-reconstructed but also sufficiently small for the manifold to have little curvature.

The LLE algorithm is best served if the graph formed by linking each point to its neighbors is connected. If the graph is not connected, the LLE algorithm can be applied separately to each of the disconnected subgraphs.

2. Constrained least-squares fits. Reconstruct $\mathbf{x}_i$ by a linear function of its K nearest neighbors,

$$\widehat{\mathbf{x}}_i = \sum_{j=1}^{n} w_{ij}\mathbf{x}_j, \tag{16.40}$$

where w_{ij} is a scalar weight for $\mathbf{x}_j$ with unit sum, $\sum_j w_{ij} = 1$, for translation invariance; if $\mathbf{x}_\ell \notin N_i^K$, then set $w_{i\ell} = 0$ in (16.40). Set $\mathbf{W} = (w_{ij})$ to be a sparse $(n \times n)$-matrix of weights (there are only nK nonzero elements). Find optimal weights $\{\widehat{w}_{ij}\}$ by solving

$$\widehat{\mathbf{W}} = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} \|\mathbf{x}_i - \sum_{j=1}^{n} w_{ij}\mathbf{x}_j\|^2, \tag{16.41}$$

subject to the *invariance constraint* $\sum_j w_{ij} = 1$, $i = 1, 2, \ldots, n$, and the *sparseness constraint* $w_{i\ell} = 0$ if $\mathbf{x}_\ell \notin N_i^K$.

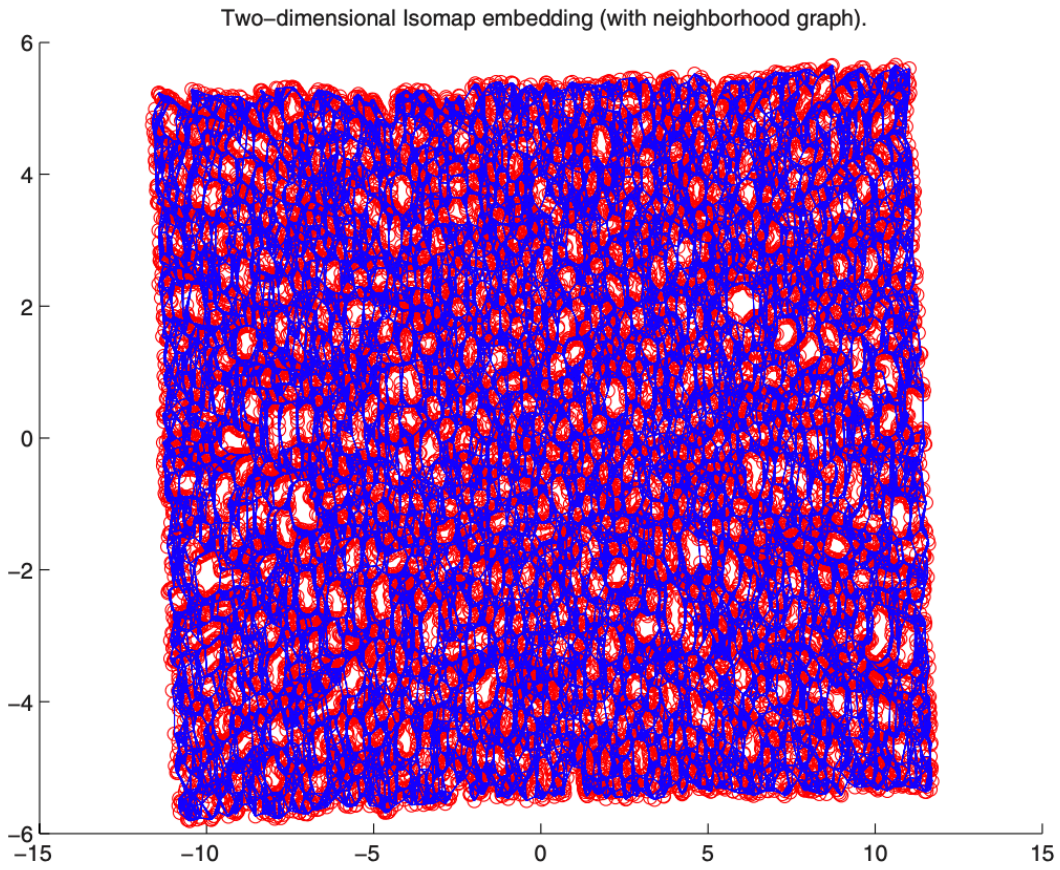


FIGURE 16.11. *Two-dimensional* LANDMARK ISOMAP *embedding, with neighborhood graph, of the complete set of* $n = 20{,}000$ *Swiss-Roll data points. The number of neighborhood points is* $K = 7$*, and the number of landmark points is* $m = 50$.

The matrix $\widehat{\mathbf{W}}$ can be obtained as follows. For a given point $\mathbf{x}_i$, the summand of (16.41) can be written as

$$\| \sum_j w_{ij}(\mathbf{x}_i - \mathbf{x}_j)\|^2 = \mathbf{w}_i^\tau \mathbf{G} \mathbf{w}_i, \tag{16.42}$$

where $\mathbf{w}_i = (w_{i1}, \cdots, w_{in})^\tau$, only K of which are non-zero, and $\mathbf{G} = (G_{jk})$, $G_{jk} = (\mathbf{x}_i - \mathbf{x}_j)^\tau (\mathbf{x}_i - \mathbf{x}_k)$, $j, k \in N_i^K$, is a symmetric, nonnegative-definite, $(n \times n)$-matrix. Using the Lagrangean multiplier μ, we minimize the function

$$f(\mathbf{w}_i) = \mathbf{w}_i^\tau \mathbf{G} \mathbf{w}_i - \mu(\mathbf{1}_n^\tau \mathbf{w}_i - 1).$$

Differentiating $f(\mathbf{w}_i)$ with respect to $\mathbf{w}_i$ and setting the result equal to zero yields $\widehat{\mathbf{w}}_i = \frac{\mu}{2} \mathbf{G}^{-1} \mathbf{1}_n$. Premultiplying this last result by $\mathbf{1}_n^\tau$ gives us the optimal weights

$$\widehat{\mathbf{w}}_i = \frac{\mathbf{G}^{-1}\mathbf{1}_n}{\mathbf{1}_n^\tau \mathbf{G}^{-1} \mathbf{1}_n},$$

where it is understood that for $\mathbf{x}_\ell \notin N_i^K$, the corresponding element, $\widehat{w}_{i\ell}$, of $\widehat{\mathbf{w}}_i$ is zero. Note that we can also write $\mathbf{G}(\frac{2}{\mu}\widehat{\mathbf{w}}_i) = \mathbf{1}_n$; so, the same result can be obtained by solving the linear system of n equations $\mathbf{G}\widehat{\mathbf{w}}_i = \mathbf{1}_n$, where any $\mathbf{x}_\ell \notin N_i^K$ has weight $\widehat{w}_{i\ell} = 0$, and then rescaling the weights to sum to one. Collect the resulting optimal weights for each data point (and all other zero-weights) into a sparse $(n \times n)$-matrix $\widehat{\mathbf{W}} = (\widehat{w}_{ij})$ having only nK nonzero elements.

3. Eigenproblem. Fix the optimal weight matrix $\widehat{\mathbf{W}}$ found at step 2. Find the $(t \times n)$-matrix $\mathbf{Y} = (\mathbf{y}_1 . \cdots , \mathbf{y}_n)$, $t \ll r$, of embedding coordinates that solves

$$\widehat{\mathbf{Y}} = \arg\min_{\mathbf{Y}} \sum_{i=1}^{n} \| \mathbf{y}_i - \sum_{j=1}^{n} \widehat{w}_{ij} \mathbf{y}_j \|^2, \tag{16.43}$$

subject to the constraints $\sum_i \mathbf{y}_i = \mathbf{Y}\mathbf{1}_n = \mathbf{0}$ and $n^{-1} \sum_i \mathbf{y}_i \mathbf{y}_i^\tau = n^{-1}\mathbf{Y}\mathbf{Y}^\tau = \mathbf{I}_t$.

These constraints are imposed to fix the translation, rotation, and scale of the embedding coordinates so that the objective function will be invariant. We can show that (10.60) can be written as

$$\widehat{\mathbf{Y}} = \arg\min_{\mathbf{Y}} \operatorname{tr}\{\mathbf{Y}\mathbf{M}\mathbf{Y}^\tau\} \tag{16.44}$$

where $\mathbf{M}$ is the sparse, symmetric, and nonnegative-definite $(n \times n)$-matrix $\mathbf{M} = (\mathbf{I}_n - \widehat{\mathbf{W}})^\tau (\mathbf{I}_n - \widehat{\mathbf{W}})$.

The objective function $\mathrm{tr}\{\mathbf{Y}\mathbf{M}\mathbf{Y}^\tau\}$ in (16.56) has a unique global minimum given by the eigenvectors corresponding to the *smallest* $t+1$ eigenvalues of $\mathbf{M}$. The smallest eigenvalue of $\mathbf{M}$ is zero with corresponding eigenvector $\mathbf{v}_n = n^{-1/2}\mathbf{1}_n$. Because the sum of coefficients of each of the other eigenvectors, which are orthogonal to $n^{-1/2}\mathbf{1}_n$, is zero, if we ignore the smallest eigenvalue (and associated eigenvector), this will constrain the embeddings to have mean zero. The optimal solution then sets the rows of the $(t \times n)$-matrix $\widehat{\mathbf{Y}}$ to be the t remaining n-dimensional eigenvectors of $\mathbf{M}$,

$$\widehat{\mathbf{Y}} = (\widehat{\mathbf{y}}_1, \ldots, \widehat{\mathbf{y}}_n) = (\mathbf{v}_{n-1}, \cdots, \mathbf{v}_{n-t})^\tau, \tag{16.45}$$

where $\mathbf{v}_{n-j}$ is the eigenvector corresponding to the $(j+1)$st smallest eigenvalue of $\mathbf{M}$. The sparseness of $\mathbf{M}$ enables eigencomputations to be carried out very efficiently.

Because LLE preserves local (rather than global) properties of the underlying manifold, it is less susceptible to introducing false connections in $\mathcal{G}$ and can successfully embed nonconvex manifolds. However, like ISOMAP, it has difficulty with manifolds that contain holes.

Homework

Please check out the the following link

https://scikit-learn.org/stable/modules/manifold.html