Other Dimension Reduction Techniques

Advanced Methods for Data Analysis (36-402/36-608)

Spring 2014

1 Classical multidimensional scaling

1.1 PCA and SVD

- Recall that last time we learned principal component analysis (PCA) applied to a data matrix $X \in \mathbb{R}^{n \times p}$ with rows $x_1, \ldots x_n \in \mathbb{R}^p$. (We always center the columns of X before the analysis, and typically also scale them to have sample variance 1.) This can be summarized with the singular value decomposition (SVD) of X:

$$\begin{array}{ccccc} X & = & U & D & V^T \\ n \times p & & n \times p & p \times p & p \times p \end{array}$$

- Here $D = \mathrm{diag}(d_1, \ldots d_p)$ is diagonal with $d_1 \geq \ldots \geq d_p \geq 0$, and U, V both have orthonormal columns. Recall that:
  - the columns of V, $v_1, \ldots v_p \in \mathbb{R}^p$, are the principal component directions
  - the columns of U, $u_1, \ldots u_p \in \mathbb{R}^n$, are the normalized principal component scores
  - the jth diagonal element of D squared and divided by n, d_j^2/n, is the variance explained by v_j
- Computing $XV = UDV^TV = UD$ tells us that $Xv_j = d_j u_j$ for every $j = 1, \ldots p$, as expected (since this is how we defined the normalized principal component scores). Therefore, to perform dimension reduction, we choose some $k < p$, and define

$$Z = XV_k = \left[\begin{array}{cccc} Xv_1 & Xv_2 & \ldots & Xv_k \end{array} \right] = \left[\begin{array}{cccc} d_1 u_1 & d_2 u_2 & \ldots & d_k u_k \end{array} \right] = U_k D_k, \quad (1)$$

where $V_k \in \mathbb{R}^{p \times k}$, $U_k \in \mathbb{R}^{n \times k}$ are the first k columns of U, V and $D_k \in \mathbb{R}^{k \times k}$ is the first k diagonal elements of D. Hence $Z \in \mathbb{R}^{n \times k}$, and the rows of Z, call them $z_1, \ldots z_n \in \mathbb{R}^k$, give us a new lower-dimensional representation for the data points $x_1, \ldots x_n \in \mathbb{R}^p$

1.2 PCA from inner products only

- Here's a question that will get us started thinking about dimension reduction in a different way: could we recover the low-dimension representation Z defined above, if we were only given the inner products $XX^T \in \mathbb{R}^{n \times n}$, rather than the data points themselves $X \in \mathbb{R}^{n \times p}$? Note that the (i, j) element of XX^T is

$$(XX^T)_{ij} = x_i^T x_j,$$

where x_i and x_j are the ith and jth data points (rows of X)

- The answer: yes! Looking at (1), we see that the lower-dimensional representation can be expressed as $Z = U_k D_k$. But from the SVD of X,

$$XX^T = UDV^T VDU^T = UD^2U^T,$$

because $V^T V = I$. The above is called an *eigendecomposition* of $X^T X$, i.e., $d_1^2, \dots d_p^2$ are the eigenvalues of XX^T and $u_1, \dots u_p$ are corresponding eigenvectors. So, what we've learned: we can compute the top k eigenvalues and eigenvectors of XX^T, and then form the product

$$Z = U_k D_k = \begin{bmatrix} u_1 & u_2 & \dots & u_k \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \dots & \\ & & & d_k \end{bmatrix},$$

which gives us the desired low-dimensional representation

1.3 PCA from distances only

- Let's push it even further: suppose that we were only given a pairwise distance matrix $\Delta \in \mathbb{R}^{n \times n}$, whose (i, j) element is defined by

$$\Delta_{ij} = \|x_i - x_j\|_2.$$

I.e., we are only given the distances between pairs of points $x_1, \dots x_n \in \mathbb{R}^p$ and not the points themselves (nor knowledge of their ambient dimension p). Could we still recover the lower-dimension representation Z defined above?

- This seems even more ambitious, but remarkably, the answer is still: yes! The trick is to recover the inner product matrix $B = XX^T$ from Δ, and then apply the arguments explained in the last section to compute Z. This total procedure is called *(classical) multidimensional scaling*, and for completeness, its steps are:

  1. Recover the inner product matrix $B = XX^T$ from Δ
  2. Compute an eigendecomposition $B = UD^2U^T$, and form $Z = U_k D_k$ where U_k contains the top k eigenvectors and D_k the (square roots) of the top k eigenvalues

- Step 2 should be clear from the last section. But how would we perform Step 1? It turns out that by letting

$$A_{ij} = -\Delta_{ij}^2, \quad i, j = 1, \dots n,$$

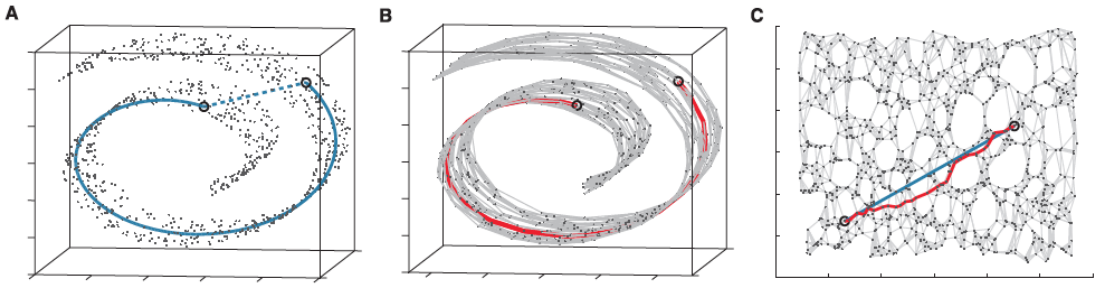
and then double centering A—i.e., centering both the rows and columns of A—we can recover the inner product matrix $B = XX^T$. (The order of centering doesn't matter, i.e., we can center the rows first, or the columns first)

- In summary, the low-dimensional representation Z that we would usually obtain from PCA on X can also be obtained from the pairwise distances Δ, and the algorithm for this is called multidimensional scaling

2 Custom distances and Isomap

- The multidimensional scaling algorithm from the last section takes as an input a pairwise distance matrix $\Delta \in \mathbb{R}^{n \times n}$, computed over some data points $x_1, \dots x_n$, and produces a low-dimensional representation $z_1, \dots z_n \in \mathbb{R}^k$ for some k. The default (classic) choice is to define $\Delta_{ij} = \|x_i - x_j\|_2$, the Euclidean distance between points $x_i, x_j \in \mathbb{R}^p$. But really, we could pass any measure of distances $\Delta_{ij} = d(x_i, x_j)$ to the multidimensional scaling algorithm, and see what comes out for the low-dimensional representation Z

- Note that by using a novel distance or metric $\Delta_{ij} = d(x_i, d_j)$, we don't just get the same principal component scores representation out of multidimensional scaling, but we get a different, novel low-dimensional representation
- Why would we want to do this? Well, using Euclidean distance $\Delta_{ij} = \|x_i - x_j\|_2$ assumes that the most appropriate way to measure the distance between x_i and x_j, in the context of our data sample $x_1, \ldots x_n$, is with a straight line. But in some cases, the straight-line distance between x_i and x_j is not may not be an ideal reflection of their discrepancy, in the context of the problem; the idea is that by defining a more appropriate distance measure, we can achieve a more appropriate low-dimensional representation through multidimensional scaling
- There are many examples of custom distances that can be appropriately defined in different problem settings; i.e., if we think of $x_1, \ldots x_n$ as images, or videos, or words, etc., there may be a different appropriate measure of distance for each of these settings
- Here we describe one general-purpose way of flexibly measuring the distances between points $x_1, \ldots x_n \in \mathbb{R}^p$, called *isometric feature mapping* or Isomap.[1] The basic idea is to construct a graph $G = (V, E)$, i.e., construct edges E between vertices $V = \{1, \ldots n\}$, based on the structure between $x_1, \ldots x_n \in \mathbb{R}^p$. Then we define a *graph distance* $\Delta_{ij} = d^{\text{graph}}(x_i, x_j)$ between x_i and x_j, and use multidimensional scaling to compute our low-dimensional representation



(From Tenenbaum et al. (2000))

- Constructing the graph: for each pair i, j, we connect i and j with an edge if either:
  - x_i is one of x_j's m nearest neighbors, or
  - x_j is one of x_i's m nearest neighbors

  The weight of this edge $e = \{i, j\}$ is then $w_e = \|x_i - x_j\|_2$
- Defining graph distances: now that we have built a graph, i.e., we have built an edge set E, we define the graph distance $\Delta_{ij} = d^{\text{graph}}(x_i, x_j)$ between x_i and x_j to be the *distance of the shortest path* in our graph from i to j. This is the minimum sum of edges weights over all paths P connecting i to j,

$$d^{\text{graph}}(x_i, x_j) = \min_{\substack{\text{paths } P \subseteq E \\ \text{from } i \text{ to } j}} \sum_{e \in P} w_e,$$

  and can be computed by standard graph algorithms from computer science (e.g., Dijkstra's algorithm or Floyd's algorithm)
- Finally we run Δ through multidimensional scaling to achieve the low dimensional representation $z_1, \ldots z_n \in \mathbb{R}^k$ for some chosen number of dimensions k

[1] Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

- Isomap works well when the sample points $x_1, \ldots x_n$ lie close to what is known as a *smooth manifold* of $\mathbb{R}^p$; you can think of this as just a lower dimensional subspace of $\mathbb{R}^p$ that has been smoothly deformed. For lots of neat examples, see the referenced Isomap paper or `http://isomap.stanford.edu`

3 Local linear embedding

- *Local linear embedding*[2] is a similar method in spirit but its details are quite different. It doesn't rely on multidimensional scaling at all. The basic idea has two steps:

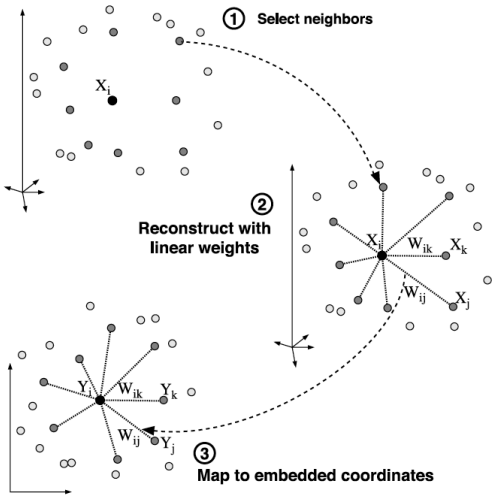
  1. Learn a bunch of local approximations to the structure between $x_1, \ldots x_n \in \mathbb{R}^p$
  2. Learn a low-dimensional representation $z_1, \ldots z_n \in \mathbb{R}^k$ whose structure best matches these local approximations

- What is meant by such local approximations? We simply try to predict each x_i by a linear function of nearby points x_j (hence the name "local linear" embedding). In particular, for each $x_i \in \mathbb{R}^p$, we first find its m nearest neighbors, and collect their indices as $\mathcal{N}(i)$. Then we build a weight vector $w_i \in \mathbb{R}^n$, setting $w_{ij} = 0$ for $j \notin \mathcal{N}(i)$ and fitting w_{ij} for $j \in \mathcal{N}(i)$ by minimizing

$$\Big\| x_i - \sum_{j \in \mathcal{N}(i)} w_{ij} x_j \Big\|_2^2$$

- Using the weights $w_1, \ldots w_n \in \mathbb{R}^n$, we fit the low-dimensional representation $z_1, \ldots z_n \in \mathbb{R}^k$, by minimizing

$$\sum_{i=1}^n \Big\| z_i - \sum_{j=1}^n w_{ij} z_j \Big\|_2^2$$



(From Roweis et al. (2000))

- For several neat examples, see the referenced paper

[2] Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"