LECTURE NOTE #11

PROF. ALAN YUILLE

1. NonLinear Dimension Reduction

Spectral Methods. The basic idea is to assume that the data lies on a manifold/surface in D-dimensional space, see figure (1) Perform multi-dimensional scaling, or other dimension reduction method, using distances calculated on the manifold, see figure (2).

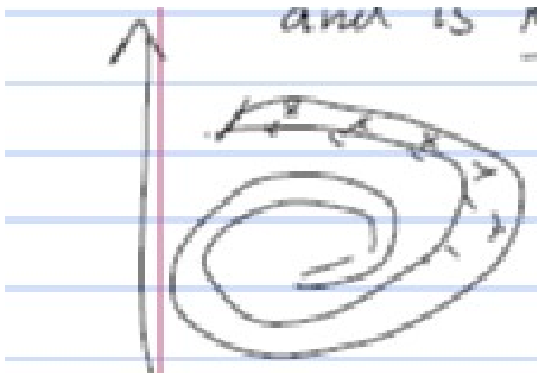

FIGURE 1. We assume that the data lies on a manifold. This is a surface which is locally flat.

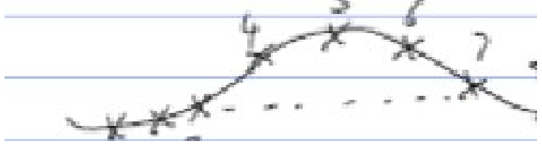

FIGURE 2. The distance is calculated on the manifold. I.e. the distance from x_3 to x_7 is the sum of the distances from x_3 to x_4, from x_4 to x_5, from x_5 to x_6, from x_6 to x_7. It is *not* the direct distance between x_3 and x_7.

Strategy: (i) define a sparse graph using kNN (nearest neighbors), (ii) derive a matrix from graph weights, (iii) derive the embedding from the graph weights.

2. Example 1: ISOMAP

Datapoints $\mathcal{X} = \{\vec{x}_i : i = 1, ..., N\}$.

Step 1. Construct an adjacency graph $(\mathcal{V}, \mathcal{E})$, where the datapoints are the vertices $\mathcal{V}$ (hence $|\mathcal{V}| = N$). The edges $\mathcal{E}$ connect each node to its k nearest neighbors, see figure (2).

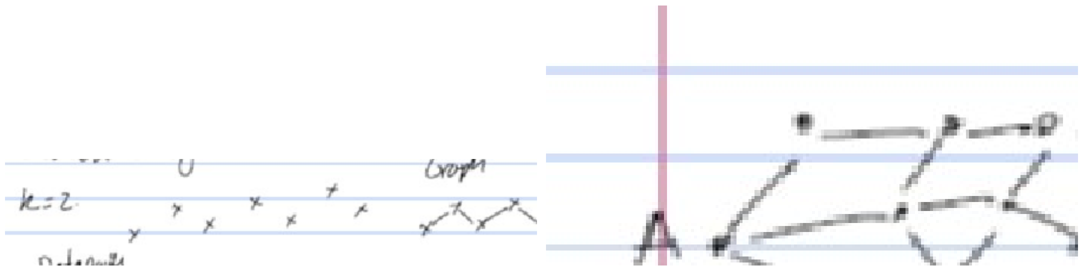

Figure 3. K-nearest neighbors (left) and dynamic programming (right) – dynamic programming is described in the lecture on probability on graphs.

Step 2. Estimate geodesics, the shortest distance between two nodes x_i, x_j of the graph on the manifold, see figure (2). Use dynamic programming to calculate the shortest path from A to B, see figure (2). This gives a geodesic distance between x_i and x_j measured on the manifold.

Step 3. Apply Multi-dimensional scaling (MDS) to the Δ_{iJ}. MDS is described at the end of this lecture.

The complexity of this algorithm is as follows. (1) k nearest neighbor – complexity $O(n^2D)$, (2) Shortest path by dynamic programming – complexity $O(n^2k + n^2 \log n)$, MDS – complexity $O(n^2d)$. The overall complexity is high. Impractical for large numbers of datapoints. Possible solution – Lamdnark ISOMAP: identify a subset of datapoints as "landmarks", do ISOMAP on the landmarks, interpolate for other datapoints.

Comment. This is a simple and intuitive algorithm. There is a problem is the algorithm finds short-cuts, see figure (2). It only projects the datapoints and does not specify how to project new data.

The results are intuitive. For example, take two images of a face and interpolate in ISOMAP feature space. ISOMAP was the first of several methods which used similar strategies.

Theoretical guarantee. If the manifold is isometric to a convex subset of Euclidean space then ISOMAP recovers this subspace (up to rotation and translation). An isometry is a distance preserving map.

FIGURE 4. Short-cuts can cause a problems for ISOMAP. The short cut is a connection between two different spirals of the surface. It prevents the geodesic from lying on the surface and instead allow it to take short cutes.

3. Local Linear Embedding (LLE)

Step 1. As for ISOMAP, construct adjacency graph using k-NN.

Step 2. Construct weights – characterize the local geometry by weights w_{ij} chosen to minimize:

$$\Phi(w) = \sum |\vec{x}_i - \sum_j w_{ij}\vec{x}_j|^2.$$

This is the error of reconstructing/predicting $\vec{x}_i$ from its neighbors. Here $w_{ii} = 0, \sum_j w_{ij} = 1$.

Local invariance. The optimal weights are invariant to rotation, translation, and scaling.

Step 3. Linearization. This maps inputs to outputs $\vec{y}_i$ to minimize the reconstruction error:

$$\Psi(\vec{y}) = \sum_i |\vec{y}_i - \sum_j w_{ij}\vec{y}_j|^2$$

With the constraint $\sum_j \vec{y}_j = 0$. And impose $1/N \sum_i \vec{y}_i \vec{y}_i^T = I$ (the identity matrix).

Reformulate:

$$\Psi(\vec{y}) = \sum_{ij} \Psi_{ij} \vec{y}_i \cdot \vec{y}_j.$$

Where $\Psi = (I - W)^T (I - W)$. Use the bottom $d + 1$ eigenvalues – where we are projecting onto d dimensions.

Both optimizations of $\Phi(w)$ and $\Psi(\vec{y})$ can be performed by linear algebra.

Properties. This has polynomial time optimization (no need to compute geodesics). It does not estimate the dimensions. It has no theoretical guarantees.

4. Laplacian Eignemaps

This is closely related to Local Linear Embedding (LLE).

Step 1. Build adjacency graph.

Step 2. Assign weights $w_{ij} = \exp\{-\beta|\vec{x}_i - \vec{x}_j|^2\}$.

Step 3. Compute outputs by minimizing:

$$\Psi(\vec{y}) = \sum_{ij} \frac{w_{ij}|\vec{y}_i - \vec{y}_j|^2}{\sqrt{D_{ii}D_{jj}}}$$

Here $D_{ii} = \sum_j w_{ij}$.

Questions. (1) How to estimate dimensionality? ISOMAP does this (by the eigenvalues) but LLE does not. (2) Should we preserve distances? Will other criteria give us better ways to reduce the dimension.

Two new algorithms. Algorithm 1 preserves local distances (isometry). Algorithm 2 preserves local angles. Conformal mappings.

An isometry is a smooth invertible mapping that preserves distances and looks locally like rotation and translation. Intuition – bend a sheet of paper but do not tear it.

Algorithm 1. Maximum Variance Unfolding. This generalizes the PCA computation of "maximum variance subspace".

Let $K_{ij} = \vec{y}_i \cdot \vec{y}_j$ be the Gram matrix.

Maximize $\sum_i K_{ii}$ subject to: (i) $K_{ii} + K_{jj} - 2K_{ij} = |\vec{x}_i - \vec{x}_j|^2$. (ii) $\sum_i K_{ii} = 0$. (iii) K is positive semi-definite.

Semi-definite programming (SDP).

Summary: (1) nearest neighbbor to compute adjacency graph. (2) Semi-definite programming (SDP) to compute the maximum variance unfolding. (3) Diagonalize the Gram matrix – estimate the dimension from the rank. Speed up by using landmarks. But is still limited to isometric transformations.

Algorithm 2, Preserve local angles. Conformal mapping: rotation, translation, and scaling.

Construct the k nearest neighbor graph. Compare edges, are lengths the same up to a scaling constant.

$$D = \sum_{ijk} \zeta_{ij}\zeta_{ik}\{|\vec{z}_i - \vec{z}_k|^2 - s_i|\vec{x}_j - \vec{x}_k|^2\}^2.$$

Given inputs $\{\vec{x}_i\}$, can we compute outputs $\{\vec{z}_i\}$ and scaling parameter s_i such that the angles are well-preserved?

Problem. There is a trivial solution – $s = 0$, $\vec{z} = 0$. Need to regularize the solution. Require the $\vec{z}$'s to be expressed in form $\vec{z}_i = L\vec{y}_i$, with $L = D - W$ and $D_{ii} = \sum_j w_{ij}$. Only use the m smallest eigenvectors of L (prevents the trivial solution $Tr(L^T L) = 1$).

Strategy to minimize $D(L, S)$ (require $\vec{z}_i = L\vec{y}_i$). Solve for the scaling factor $\partial D/\partial s = 0$. This reduces to a semi-definite programming problem.

5. Relation to Kernel PcA

kPCA (kernel PCA) is linear in feature space and non-linear in input space.

But can kernel PCA perform isometric transformations?

Answer – No, although kPCA with RBF kernel is approximately a local isometry.

But spectral methods – e.g., ISOMAP – can be seen as ways to construct kernel matrices. I.e. not generic kernels, but data-driven kernels.

6. Multidimensional scaling

MDS is a linear projection method similar to PCA. But it has the attractive property that it does not require knowing the features $\vec{x}$ of the datapoints. Instead it only needs to know the distances between two datapoints. This is attractive for problems where it is hard to decide what features to use – e.g., for representing a picture– but easier to decide if two pictures are similar. This also makes it suitable for nonlinear dimension reduction because MDS depends on the distance on the manifold.

The basic idea of MDS is to project the data to preserve the distances $|\vec{x}_i - \vec{x}_j|$ between the datapoints. In other words, we project $\vec{x}$ to a lower-dimensional vector $\vec{y}$ so that $|\vec{y}_i - \vec{y}_j| \approx |\vec{x}_i - \vec{x}_j|$.

This projection constraint is imposed on the dot products $\vec{x}_i \cdot \vec{x}_j \approx \vec{y}_i \cdot \vec{y}_j$, which will imply the result.

Key Result: we only need to know $\Delta_{ij} = |\vec{x}_i - \vec{x}_j|$ in order to calculate $\vec{x}_i \cdot \vec{x}_j$ (i.e. we only need to know the distances between points and not their positions $\vec{x}$).

Result.

$$\vec{x}_i \cdot \vec{x}_j = \frac{1}{2N} \sum_k \Delta_{ik}^2 + \frac{1}{2N} \sum_l \Delta_{lj}^2 - \frac{1}{2N^2} \sum_{lk} \Delta_{lk}^2 - \frac{1}{2} \Delta_{ij}^2,$$

provided $\sum_i \vec{x}_i = 0$ (which we can always enforce).

Proof. $|Delta_{ij}^2 = |\vec{x}_i|^2 + |\vec{x}_j|^2 - 2\vec{x}_i \cdot \vec{x}_j$. Let $T = \sum_i |\vec{x}_i|^2$. (Note that $\sum_i \vec{x}_i \cdot \vec{x}_j = 0 = \sum_j \vec{x}_i \cdot \vec{x}_j$). Then $\Delta_{ik}^2 = N|\vec{x}_i|^2 + T$, $\sum_l \Delta_{lj}^2 N|\vec{x}_j|^2 + T$, $\sum_{lk} \Delta_{lk}^2 = 2NT$. The result follows by subtraction.

Now we proceed by defining the Gram matrix:

$$G_{ij} = \vec{x}_i \cdot \vec{x}_j = \sum_k \Delta_{ik}^2 + \frac{1}{2N} \sum_l \Delta_{lj}^2 - \frac{1}{2N^2} \sum_{lk} \Delta_{lk}^2 - \frac{1}{2} \Delta_{ij^2}.$$

Define an error function for projection:

$$Err(y) = \sum_{ij} (G_{ij} - \vec{y}_i \cdot \vec{y}_j)^2.$$

Here $\vec{x}$ lies in D-dimensional space, G is an $N \times N$ matrix (N is the number of datapoints), $\vec{y}$ lies in a d-dimensional space, with $d << D$ and $d < N$.

7. MDS Procedure

To minimize $Err(y)$ we do spectral decomposition of the Gram matrix:

$$G = \sum_a \lambda_a \vec{v}^a \vec{v}^a$$

In coordinates: $G_{ij} = \sum_a \lambda_a v_i^a v_j^a$, where λ_a is the a^{th} eigenvalue of G, $\vec{v}^a$ its eigenvector, and v_i^a is the i^{th} component of the eigenvector $\vec{v}^a$. We use the convention that $\lambda_1 \geq ... \geq \lambda_n$ and, as always, the eigenvectors are orthogonal $\vec{v}^a \cdot \vec{v}^b = 0$ if $a \neq b$ ($|\vec{v}^a| = 1$).

Result. The optimal minimization of $Err(y)$ is obtained by setting the vectors $\vec{y}_i$ to be have N components $y_a^i = \sqrt{\lambda_a} v_i^a$. I.e. $\vec{y}_i = (\sqrt{\lambda_1} v_1^1, ..., \sqrt{\lambda_N} v_i^N)$.

Proof. $\vec{y}_i \cdot \vec{y}_j = \sum_a \sqrt{\lambda_a} v_i^a \sqrt{\lambda_a} v_j^a = \sum_a \lambda_a \vec{v}^a \vec{v}^a = G_{ij}$. I.e. we get $Err(y) = 0$ if we express $y_a^i = \sqrt{\lambda_a} v_i^a$.

But this does not reduce the dimension (even if it gives zero error).

We can reduce the dimension, while raising the error, by only keeping the first d components. I.e. set $\vec{y}_i = (\sqrt{\lambda_1} v_1^1, ..., \sqrt{\lambda_d} v_i^d)$ (recall that we ordered the eigenvalues $\lambda_1 \geq ... \geq \lambda_N$, so we are picking the biggest d eigenvalues).

In this case, $G_{ij} \neq \vec{y}_i \cdot \vec{y}_j^T$. Instead, we compute that

$$G_{ij} - \vec{y}_i \cdot \vec{y}_j = \sum_{a=d+1}^{N} \lambda_a v_i^a v_j^a$$

It follows that the error $Err(y) = \sum_{a=d+1}^{N} \lambda_a^2$. This justifies keeping the first d components (corresponding to the biggest d eigenvalues).

8. Relations between PCA and MDS

Both are linear projections. Both involve projections specified by the biggest d eigenvalues/eigenvectors of a matrix which depends on the data.

PCA uses the eigenvalues/eigenvectors of the covariance $\sum_{i=1}^{N} \vec{x}_i \vec{x}_i^T$. MDS uses eigenvalues/eigenvectors of the Gram matrix $\sum \vec{x}_i \cdot \vec{x}_j$.

Result. The covariance and the Gram matrix have the same eigenvalues and their eigenvectors are related.

Proof. This result was discussed in an earlier lecture (in the PCA lecture in the section which discussed SVD). To obtain it, we defined the matrix X with components x_{ia}. Then the covariance is expressed as the matrix XX^T and the Gram matrix as $X_T X$. If e is an eigenvector of XX^T with eigenvalue λ – i.e. $XX^T e = \lambda e$ – then $X^T e$ is an eigenvector of $X^T X$ with eigenvalue λ – because $X^T X X^T e = X^T (XX^T e) = X^T \lambda e = \lambda X^T e$. Similarly we can relate the eigenvectors/eigenvalues of $X^T X$ to those of XX^T.

In summary, the error criteria of PCA and MDS depend on the same eigenvalues and dimension reduction occurs by rejecting the coordinate directions corresponding to the smallest eigenvalues. But the difference is the PCA and MDS project using the eigenvectors of different, but related, matrix (correlation or Gram).