Nonlinear Dimensionality Reduction

Piyush Rai

CS5350/6350: Machine Learning

October 25, 2011
Recap: Linear Dimensionality Reduction

- Linear Dimensionality Reduction: Based on a linear projection of the data
- Assumes that the data lives close to a lower dimensional linear subspace

The data is projected on to that subspace

Data $\mathbf{X}$ is $N \times D$, Projection Matrix $\mathbf{U}$ is $D \times K$, Projection $\mathbf{Z}$ is $N \times K$

$$\mathbf{Z} = \mathbf{XU}$$

Using $\mathbf{UU}^\top = \mathbf{I}$ (orthonormality of eigenvectors), we have:

$$\mathbf{X} = \mathbf{ZU}^\top$$

Linear dimensionality reduction does a matrix factorization of $\mathbf{X}$
Matrix Factorization view helps reveal latent aspects about the data.

In PCA, each principal component corresponds to a latent aspect.
Examples: Netflix Movie-Ratings Data

- $K$ principal components corresponds to $K$ underlying genres
- $Z$ denotes the extent each user likes different movie genres
Examples: Amazon Book-Ratings Data

- $K$ principal components corresponds to $K$ underlying genres
- $Z$ denotes the extent each user likes different book genres
Examples: Identifying Topics in Document Collections

- $K$ principal components corresponds to $K$ underlying topics.
- $Z$ denotes the extent each topic is represented in a document.
$K$ principal components corresponds to $K$ image templates (dictionary)

$Z$ denotes the extent each dictionary element is represented in an image
Nonlinear Dimensionality Reduction

- Given: Low-dim. surface embedded \textit{nonlinearly} in high-dim. space
  - Such a structure is called a \textbf{Manifold}

- Goal: Recover the low-dimensional surface
Linear Projection may not be good enough..

Consider the swiss-roll dataset (points lying close to a manifold)

- Linear projection methods (e.g., PCA) can't capture intrinsic nonlinearities
Nonlinear Dimensionality Reduction

- We want to do nonlinear projections.
- Different criteria could be used for such projections.
- Most nonlinear methods try to preserve the neighborhood information:
  - Locally linear structures (locally linear $\Rightarrow$ globally nonlinear)
  - Pairwise distances (along the nonlinear manifold)
- Roughly translates to “unrolling” the manifold.
Nonlinear Dimensionality Reduction

Two ways of doing it:

- **Nonlinearize** a linear dimensionality reduction method. E.g.:
  - **Kernel PCA** (nonlinear PCA)

- Using **manifold based methods**. E.g.:
  - **Locally Linear Embedding (LLE)**
  - **Isomap**
  - Maximum Variance Unfolding
  - **Laplacian Eigenmaps**
  - And several others (Hessian LLE, Hessian Eigenmaps, etc.)
Kernel PCA

- Given $N$ observations $\{x_1, \ldots, x_N\}$, $\forall x_n \in \mathbb{R}^D$, define the $D \times D$ covariance matrix (assuming centered data $\sum_n x_n = 0$)

$$S = \frac{1}{N} \sum_{n=1}^{N} x_n x_n^T$$

- Linear PCA: Compute eigenvectors $u_i$ satisfying: $Su_i = \lambda_i u_i \; \forall i = 1, \ldots, D$

- Consider a nonlinear transformation $\phi(x)$ of $x$ into an $M$ dimensional space

- $M \times M$ covariance matrix in this space (assume centered data $\sum_n \phi(x_n) = 0$)

$$C = \frac{1}{N} \sum_{n=1}^{N} \phi(x_n) \phi(x_n)^T$$

- Kernel PCA: Compute eigenvectors $v_i$ satisfying: $Cv_i = \lambda_i v_i \; \forall i = 1, \ldots, M$

- Ideally, we would like to do this without having to compute the $\phi(x_n)$'s
Kernel PCA

- **Kernel PCA**: Compute eigenvectors $v_i$ satisfying: $Cv_i = \lambda_i v_i$
- Plugging in the expression for $C$, we have the eigenvector equation:

$$\frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\{\phi(x_n)^\top v_i\} = \lambda_i v_i$$

- Using the above, we can write $v_i$ as: $v_i = \sum_{n=1}^{N} a_{in} \phi(x_n)$
- Plugging this back in the eigenvector equation:

$$\frac{1}{N} \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^\top \sum_{m=1}^{N} a_{im} \phi(x_m) = \lambda_i \sum_{n=1}^{N} a_{in} \phi(x_n)$$

- Pre-multiplying both sides by $\phi(x_l)^\top$ and re-arranging

$$\frac{1}{N} \sum_{n=1}^{N} \phi(x_l)^\top \phi(x_n) \sum_{m=1}^{N} a_{im} \phi(x_n)^\top \phi(x_m) = \lambda_i \sum_{n=1}^{N} a_{in} \phi(x_l)^\top \phi(x_n)$$
Kernel PCA

Using $\phi(x_n)^T \phi(x_m) = k(x_n, x_m)$, the eigenvector equation becomes:

$$\frac{1}{N} \sum_{n=1}^{N} k(x_l, x_n) \sum_{m=1}^{N} a_{im} k(x_n, x_m) = \lambda_i \sum_{n=1}^{N} a_{in} k(x_l, x_n)$$

Define $K$ as the $N \times N$ kernel matrix with $K_{nm} = k(x_n, x_m)$
- $K$ is the similarity of two examples $x_n$ and $x_m$ in the $\phi$ space
- $\phi$ is implicitly defined by kernel function $k$ (which can be, e.g., RBF kernel)

Define $a_i$ as the $N \times 1$ vector with elements $a_{in}$

Using $K$ and $a_i$, the eigenvector equation becomes:

$$K^2 a_i = \lambda_i N K a_i \quad \Rightarrow \quad Ka_i = \lambda_i N a_i$$

This corresponds to the original Kernel PCA eigenvalue problem $C v_i = \lambda_i v_i$

For a projection to $K < D$ dimensions, top $K$ eigenvectors of $K$ are used.
Kernel PCA: Centering the Data

- In PCA, we centered the data before computing the covariance matrix.
- For kernel PCA, we need to do the same.

\[ \tilde{\phi}(x_n) = \phi(x_n) - \frac{1}{N} \sum_{l=1}^{N} \phi(x_l) \]

- How does it affect the kernel matrix \( K \) which is eigen-decomposed?

\[ \tilde{K}_{nm} = \tilde{\phi}(x_n)^\top \tilde{\phi}(x_m) \]
\[ = \phi(x_n)^\top \phi(x_m) - \frac{1}{N} \sum_{l=1}^{N} \phi(x_n)^\top \phi(x_l) - \frac{1}{N} \sum_{l=1}^{N} \phi(x_l)^\top \phi(x_m) + \frac{1}{N^2} \sum_{j=1}^{N} \sum_{l=1}^{N} \phi(x_j)^\top \phi(x_l) \]
\[ = k(x_n, x_m) - \frac{1}{N} \sum_{l=1}^{N} k(x_n, x_l) - \frac{1}{N} \sum_{l=1}^{N} k(x_l, x_m) + \frac{1}{N^2} \sum_{j=1}^{N} \sum_{l=1}^{N} k(x_l, x_l) \]

- In matrix notation, the centered \( \tilde{K} = K - \mathbf{1}_N K - K \mathbf{1}_N + \mathbf{1}_N K \mathbf{1}_N \)
- \( \mathbf{1}_N \) is the \( N \times N \) matrix with every element = \( 1/N \)
- Eigen-decomposition is then done for the centered kernel matrix \( \tilde{K} \)
Suppose \{a_1, \ldots, a_K\} are the top \(K\) eigenvectors of kernel matrix \(\tilde{\mathbf{K}}\).

The \(K\)-dimensional KPCA projection \(\mathbf{z} = [z_1, \ldots, z_K]\) of a point \(\mathbf{x}\):

\[
z_i = \phi(\mathbf{x})^\top \mathbf{v}_i
\]

Recall the definition of \(\mathbf{v}_i\):

\[
\mathbf{v}_i = \sum_{n=1}^{N} a_{in} \phi(\mathbf{x}_n)
\]

Thus

\[
z_i = \phi(\mathbf{x})^\top \mathbf{v}_i = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n)
\]
Manifold Based Methods

- **Locally Linear Embedding (LLE)**

- **Isomap**

- **Maximum Variance Unfolding**

- **Laplacian Eigenmaps**

- And several others (Hessian LLE, Hessian Eigenmaps, etc.)
Locally Linear Embedding

- Based on a simple geometric intuition of local linearity

- Assume each example and its neighbors lie on or close to a locally linear patch of the manifold

- LLE assumption: Projection should preserve the neighborhood
  - Projected point should have the same neighborhood as the original point
Locally Linear Embedding: The Algorithm

- Given $D$ dim. data $\{x_1, \ldots, x_N\}$, compute $K$ dim. projections $\{z_1, \ldots, z_N\}$
- For each example $x_i$, find its $L$ nearest neighbors
- Assume $x_i$ to be a weighted linear combination of the $L$ nearest neighbors
  \[ x_i \approx \sum_{j \in \mathcal{N}_i} W_{ij} x_j \]  
  (so the data is assumed locally linear)

- Find the weights by solving the following least-squares problem:
  \[
  W = \arg \min_W \sum_{i=1}^{N} \left\| x_i - \sum_{j \in \mathcal{N}_i} W_{ij} x_j \right\|^2 
  \text{ s.t. } \forall i \sum_j W_{ij} = 1
  \]

- $\mathcal{N}_i$ are the $L$ nearest neighbors of $x_i$ (note: should choose $L \geq K + 1$)
- Use $W$ to compute low dim. projections $Z = \{z_1, \ldots, z_N\}$ by solving:
  \[
  Z = \arg \min_Z \sum_{i=1}^{N} \left\| z_i - \sum_{j \in \mathcal{N}_i} W_{ij} z_j \right\|^2 
  \text{ s.t. } \forall i \sum_{i=1}^{N} z_i = 0, \quad \frac{1}{N} ZZ^\top = 1
  \]

- Refer to the LLE reading (appendix A and B) for the details of these steps

(CS5350/6350) Nonlinear Dimensionality Reduction October 25, 2011
Isometric Feature Mapping (Isomap)

A graph based algorithm based on constructing a matrix of geodesic distances

- Identify the \( L \) nearest neighbors for each data point (just like LLE)
- Connect each point to all its neighbors (an edge for each neighbor)
- Assign weight to each edge based on the Euclidean distance
- Estimate the geodesic distance \( d_{ij} \) between any two data points \( i \) and \( j \)
  - Approximated by the sum of arc lengths along the shortest path between \( i \) and \( j \) in the graph (can be computed using Djikstras algorithm)

- Construct the \( N \times N \) distance matrix \( D = \{d_{ij}^2\} \)
Isomap (Contd.)

- Use the distance matrix $D$ to construct the Gram Matrix

$$G = -\frac{1}{2}HDH$$

where $G$ is $N \times N$ and

$$H = I - \frac{1}{N}11^\top$$

$I$ is $N \times N$ identity matrix, $1$ is $N \times 1$ vector of 1s

- Do an eigen decomposition of $G$
- Let the eigenvectors be $\{v_1, \ldots, v_N\}$ with eigenvalues $\{\lambda_1, \ldots, \lambda_N\}$
  - Each eigenvector $v_i$ is $N$-dimensional: $v_i = [v_{1i}, v_{2i}, \ldots, v_{Ni}]$
- Take the top $K$ eigenvalue/eigenvectors
- The $K$ dimensional embedding $z_i = [z_{i1}, z_{i2}, \ldots, z_{iK}]$ of a point $x_i$:

$$z_{ik} = \sqrt{\lambda_k}v_{ki}$$
Isomap: Example

Digit images projected down to 2 dimensions
Isomap: Example

Face images with varying poses