Nonlinear Dimensionality Reduction

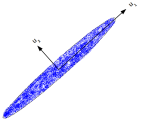
Piyush Rai

CS5350/6350: Machine Learning

October 25, 2011

Recap: Linear Dimensionality Reduction

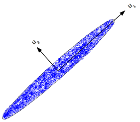
- Linear Dimensionality Reduction: Based on a linear projection of the data
- Assumes that the data lives close to a lower dimensional linear subspace



- The data is projected on to that subspace

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

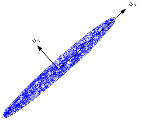


- The data is projected on to that subspace
- Data $\mathbf{X}$ is $N \times D$, Projection Matrix $\mathbf{U}$ is $D \times K$, Projection $\mathbf{Z}$ is $N \times K$

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

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 $N \times K$

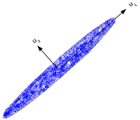
$$\mathbf{Z} = \mathbf{X}\mathbf{U}$$

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

$$\mathbf{X} = \mathbf{Z}\mathbf{U}^\top$$

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 $N \times K$

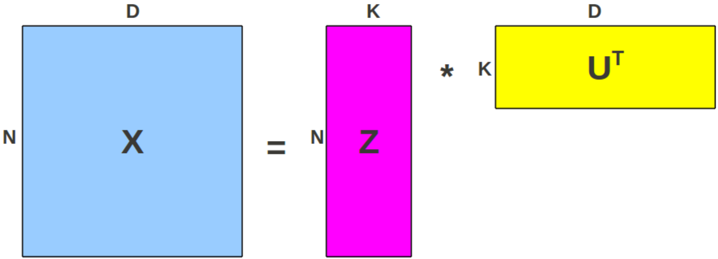
$$\mathbf{Z} = \mathbf{X}\mathbf{U}$$

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

$$\mathbf{X} = \mathbf{Z}\mathbf{U}^\top$$

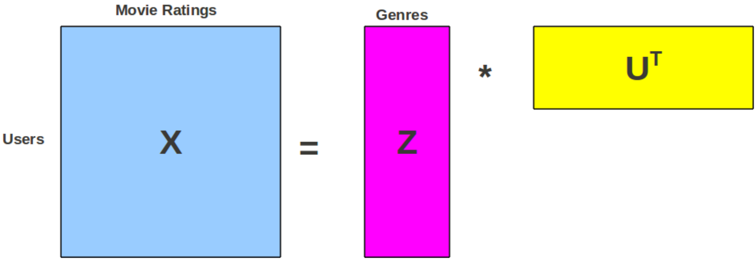
- Linear dimensionality reduction does a matrix factorization of $\mathbf{X}$

Dimensionality Reduction as Matrix Factorization



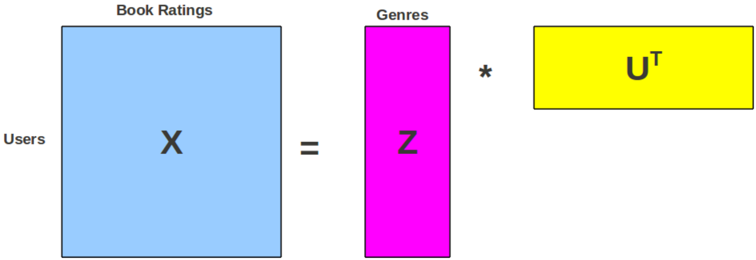
- Matrix Factorization view helps reveal latent aspects about the data
  - In PCA, each principal component corresponds to a latent aspect

Examples: Netflix Movie-Ratings Data



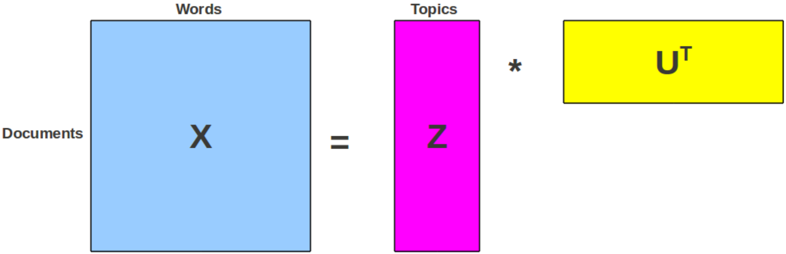
- K principal components corresponds to K underlying genres
- $\mathbf{Z}$ denotes the extent each user likes different movie genres

Examples: Amazon Book-Ratings Data



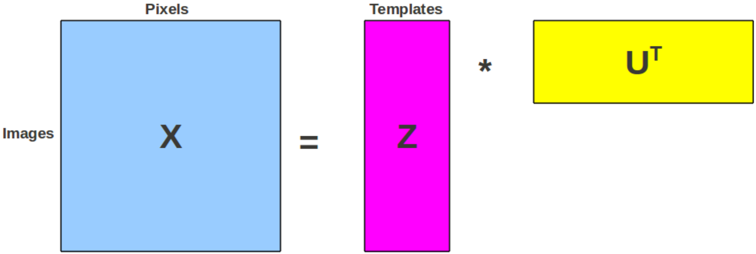
- K principal components corresponds to K underlying genres
- $\mathbf{Z}$ denotes the extent each user likes different book genres

Examples: Identifying Topics in Document Collections



- K principal components corresponds to K underlying topics
- $\mathbf{Z}$ denotes the extent each topic is represented in a document

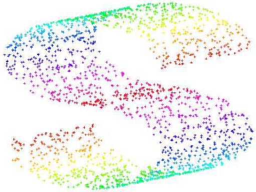
Examples: Image Dictionary (Template) Learning

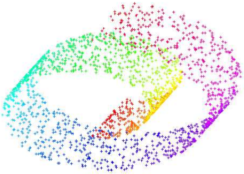


- K principal components corresponds to K image templates (dictionary)
- $\mathbf{Z}$ denotes the extent each dictionary element is represented in an image

Nonlinear Dimensionality Reduction

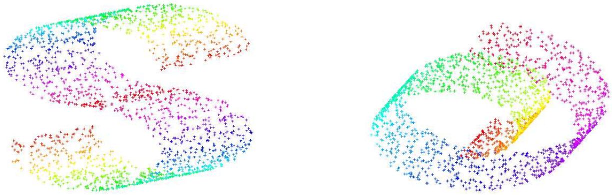
- Given: Low-dim. surface embedded **nonlinearly** in high-dim. space
  - Such a structure is called a **Manifold**



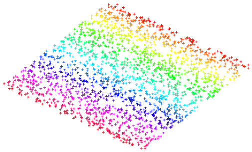


Nonlinear Dimensionality Reduction

- Given: Low-dim. surface embedded **nonlinearly** in high-dim. space
  - Such a structure is called a **Manifold**

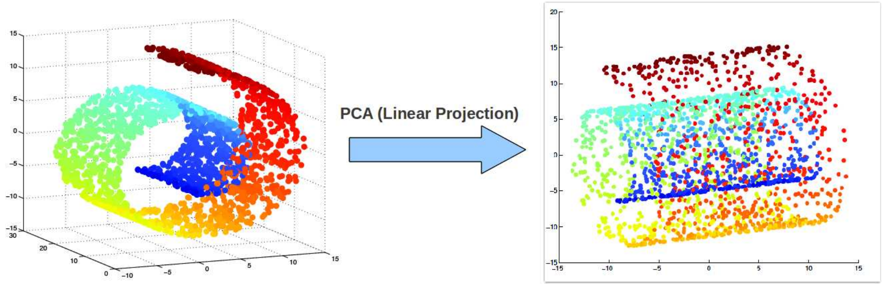


- Goal: Recover the low-dimensional surface



Linear Projection may not be good enough..

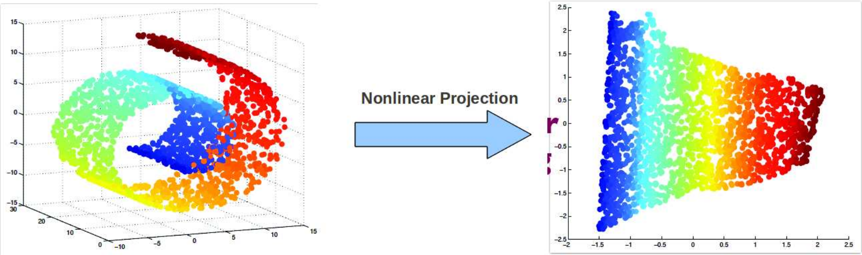
- Consider the swiss-roll dataset (points lying close to a manifold)



- 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:

Nonlinear Dimensionality Reduction

Two ways of doing it:

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

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 $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\forall \mathbf{x}_n \in \mathbb{R}^D$, define the $D \times D$ covariance matrix (assuming centered data $\sum_n \mathbf{x}_n = \mathbf{0}$)

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top$$

- Linear PCA: Compute eigenvectors $\mathbf{u}_i$ satisfying: $\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ $\forall i = 1, \ldots, D$

Kernel PCA

- Given N observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\forall \mathbf{x}_n \in \mathbb{R}^D$, define the $D \times D$ covariance matrix (assuming centered data $\sum_n \mathbf{x}_n = \mathbf{0}$)

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top$$

- Linear PCA: Compute eigenvectors $\mathbf{u}_i$ satisfying: $\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \; \forall i = 1, \ldots, D$
- Consider a nonlinear transformation $\phi(\mathbf{x})$ of $\mathbf{x}$ into an M dimensional space

Kernel PCA

- Given N observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\forall \mathbf{x}_n \in \mathbb{R}^D$, define the $D \times D$ covariance matrix (assuming centered data $\sum_n \mathbf{x}_n = \mathbf{0}$)

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top$$

- Linear PCA: Compute eigenvectors $\mathbf{u}_i$ satisfying: $\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \; \forall i = 1, \ldots, D$
- Consider a nonlinear transformation $\phi(\mathbf{x})$ of $\mathbf{x}$ into an M dimensional space
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\mathbf{x}_n) = \mathbf{0}$)

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^\top$$

Kernel PCA

- Given N observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\forall \mathbf{x}_n \in \mathbb{R}^D$, define the $D \times D$ covariance matrix (assuming centered data $\sum_n \mathbf{x}_n = \mathbf{0}$)

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top$$

- Linear PCA: Compute eigenvectors $\mathbf{u}_i$ satisfying: $\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \ \forall i = 1, \ldots, D$
- Consider a nonlinear transformation $\phi(\mathbf{x})$ of $\mathbf{x}$ into an M dimensional space
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\mathbf{x}_n) = \mathbf{0}$)

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^\top$$

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \ \forall i = 1, \ldots, M$

Kernel PCA

- Given N observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\forall \mathbf{x}_n \in \mathbb{R}^D$, define the $D \times D$ covariance matrix (assuming centered data $\sum_n \mathbf{x}_n = \mathbf{0}$)

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top$$

- Linear PCA: Compute eigenvectors $\mathbf{u}_i$ satisfying: $\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \; \forall i = 1, \ldots, D$
- Consider a nonlinear transformation $\phi(\mathbf{x})$ of $\mathbf{x}$ into an M dimensional space
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\mathbf{x}_n) = \mathbf{0}$)

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^\top$$

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \; \forall i = 1, \ldots, M$
- Ideally, we would like to do this without having to compute the $\phi(\mathbf{x}_n)$'s

Kernel PCA

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$

Kernel PCA

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$
- Plugging in the expression for $\mathbf{C}$, we have the eigenvector equation:

$$\frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\{\phi(\mathbf{x}_n)^\top \mathbf{v}_i\} = \lambda_i \mathbf{v}_i$$

Kernel PCA

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$
- Plugging in the expression for $\mathbf{C}$, we have the eigenvector equation:

$$\frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\{\phi(\mathbf{x}_n)^\top \mathbf{v}_i\} = \lambda_i \mathbf{v}_i$$

- Using the above, we can write $\mathbf{v}_i$ as: $\mathbf{v}_i = \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n)$

Kernel PCA

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$
- Plugging in the expression for $\mathbf{C}$, we have the eigenvector equation:

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\{\phi(\mathbf{x}_n)^\top \mathbf{v}_i\} = \lambda_i \mathbf{v}_i$$

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

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^\top \sum_{m=1}^{N} a_{im}\phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n)$$

Kernel PCA

- Kernel PCA: Compute eigenvectors $\mathbf{v}_i$ satisfying: $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$
- Plugging in the expression for $\mathbf{C}$, we have the eigenvector equation:

$$\frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\{\phi(\mathbf{x}_n)^\top \mathbf{v}_i\} = \lambda_i \mathbf{v}_i$$

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

$$\frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^\top \sum_{m=1}^{N} a_{im}\phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n)$$

- Pre-multiplying both sides by $\phi(\mathbf{x}_l)^\top$ and re-arranging

$$\frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_n) \sum_{m=1}^{N} a_{im}\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_n)$$

Kernel PCA

- Using $\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$, the eigenvector equation becomes:

$$\frac{1}{N} \sum_{n=1}^{N} k(\mathbf{x}_l, \mathbf{x}_n) \sum_{m=1}^{N} a_{im} k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in} k(\mathbf{x}_l, \mathbf{x}_n)$$

Kernel PCA

- Using $\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$, the eigenvector equation becomes:

$$\frac{1}{N} \sum_{n=1}^{N} k(\mathbf{x}_l, \mathbf{x}_n) \sum_{m=1}^{N} a_{im} k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in} k(\mathbf{x}_l, \mathbf{x}_n)$$

- Define $\mathbf{K}$ as the $N \times N$ **kernel matrix** with $K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$
  - $\mathbf{K}$ is the similarity of two examples $\mathbf{x}_n$ and $\mathbf{x}_m$ in the ϕ space
  - ϕ is implicitly defined by kernel function k (which can be, e.g., RBF kernel)
- Define $\mathbf{a}_i$ as the $N \times 1$ vector with elements a_{in}

Kernel PCA

- Using $\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$, the eigenvector equation becomes:

$$\frac{1}{N} \sum_{n=1}^{N} k(\mathbf{x}_l, \mathbf{x}_n) \sum_{m=1}^{N} a_{im} k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in} k(\mathbf{x}_l, \mathbf{x}_n)$$

- Define $\mathbf{K}$ as the $N \times N$ **kernel matrix** with $K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$
  - $\mathbf{K}$ is the similarity of two examples $\mathbf{x}_n$ and $\mathbf{x}_m$ in the ϕ space
  - ϕ is implicitly defined by kernel function k (which can be, e.g., RBF kernel)
- Define $\mathbf{a}_i$ as the $N \times 1$ vector with elements a_{in}
- Using $\mathbf{K}$ and $\mathbf{a}_i$, the eigenvector equation becomes:

$$\mathbf{K}^2 \mathbf{a}_i = \lambda_i N \mathbf{K} \mathbf{a}_i \quad \Rightarrow \quad \boxed{\mathbf{K}\mathbf{a}_i = \lambda_i N \mathbf{a}_i}$$

Kernel PCA

- Using $\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$, the eigenvector equation becomes:

$$\frac{1}{N} \sum_{n=1}^{N} k(\mathbf{x}_l, \mathbf{x}_n) \sum_{m=1}^{N} a_{im} k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in} k(\mathbf{x}_l, \mathbf{x}_n)$$

- Define $\mathbf{K}$ as the $N \times N$ **kernel matrix** with $K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$
  - $\mathbf{K}$ is the similarity of two examples $\mathbf{x}_n$ and $\mathbf{x}_m$ in the ϕ space
  - ϕ is implicitly defined by kernel function k (which can be, e.g., RBF kernel)
- Define $\mathbf{a}_i$ as the $N \times 1$ vector with elements a_{in}
- Using $\mathbf{K}$ and $\mathbf{a}_i$, the eigenvector equation becomes:

$$\mathbf{K}^2 \mathbf{a}_i = \lambda_i N \mathbf{K} \mathbf{a}_i \quad \Rightarrow \quad \boxed{\mathbf{K}\mathbf{a}_i = \lambda_i N \mathbf{a}_i}$$

- This corresponds to the original Kernel PCA eigenvalue problem $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$
- For a projection to $K < D$ dimensions, top K eigenvectors of $\mathbf{K}$ 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

$$\tilde{\phi}(\mathbf{x}_n) = \phi(\mathbf{x}_n) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)$$

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

$$\tilde{\phi}(\mathbf{x}_n) = \phi(\mathbf{x}_n) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)$$

- How does it affect the kernel matrix $\mathbf{K}$ which is eigen-decomposed?

$$\begin{aligned}\tilde{K}_{nm} &= \tilde{\phi}(\mathbf{x}_n)^\top\tilde{\phi}(\mathbf{x}_m)\\ &= \phi(\mathbf{x}_n)^\top\phi(\mathbf{x}_m) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_n)^\top\phi(\mathbf{x}_l) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)^\top\phi(\mathbf{x}_m) + \frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}\phi(\mathbf{x}_j)^\top\phi(\mathbf{x}_l)\end{aligned}$$

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

$$\tilde{\phi}(\mathbf{x}_n) = \phi(\mathbf{x}_n) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)$$

- How does it affect the kernel matrix **K** which is eigen-decomposed?

$$\begin{aligned}
\tilde{K}_{nm} &= \tilde{\phi}(\mathbf{x}_n)^\top \tilde{\phi}(\mathbf{x}_m) \\
&= \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_n)^\top\phi(\mathbf{x}_l) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)^\top\phi(\mathbf{x}_m) + \frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}\phi(\mathbf{x}_j)^\top\phi(\mathbf{x}_l) \\
&= k(\mathbf{x}_n,\mathbf{x}_m) - \frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_n,\mathbf{x}_l) - \frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_l,\mathbf{x}_m) + \frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}k(\mathbf{x}_l,\mathbf{x}_l)
\end{aligned}$$

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

$$\tilde{\phi}(\mathbf{x}_n) = \phi(\mathbf{x}_n) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)$$

- How does it affect the kernel matrix $\mathbf{K}$ which is eigen-decomposed?

$$\begin{aligned}
\tilde{K}_{nm} &= \tilde{\phi}(\mathbf{x}_n)^\top \tilde{\phi}(\mathbf{x}_m) \\
&= \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_l) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_m) + \frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}\phi(\mathbf{x}_j)^\top \phi(\mathbf{x}_l) \\
&= k(\mathbf{x}_n, \mathbf{x}_m) - \frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_n, \mathbf{x}_l) - \frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_l, \mathbf{x}_m) + \frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}k(\mathbf{x}_l, \mathbf{x}_l)
\end{aligned}$$

- In matrix notation, the centered $\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N$
- $\mathbf{1}_N$ is the $N \times N$ matrix with every element $= 1/N$
- Eigen-decomposition is then done for the centered kernel matrix $\tilde{\mathbf{K}}$

Kernel PCA: The Projection

- Suppose $\{\mathbf{a}_1, \ldots, \mathbf{a}_K\}$ 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 $\mathbf{x}$:

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

Kernel PCA: The Projection

- Suppose $\{\mathbf{a}_1, \ldots, \mathbf{a}_K\}$ 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 $\mathbf{v}_i$

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

Kernel PCA: The Projection

- Suppose $\{\mathbf{a}_1, \ldots, \mathbf{a}_K\}$ 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

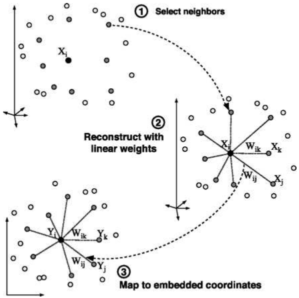
$$z_i = \phi(\mathbf{x})^\top \mathbf{v}_i = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n)$$

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 $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, compute K dim. projections $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$
- For each example $\mathbf{x}_i$, find its L nearest neighbors
- Assume $\mathbf{x}_i$ to be a weighted linear combination of the L nearest neighbors

$$\mathbf{x}_i \approx \sum_{j \in \mathcal{N}} W_{ij}\mathbf{x}_j \qquad \text{(so the data is assumed locally linear)}$$

- Find the weights by solving the following least-squares problem:

$$\boxed{W = \arg\min_W \sum_{i=1}^{N} ||\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} W_{ij}\mathbf{x}_j||^2} \qquad s.t. \forall i \quad \sum_j W_{ij} = 1$$

- $\mathcal{N}_i$ are the L nearest neighbors of $\mathbf{x}_i$ (note: should choose $L \geq K + 1$)

Locally Linear Embedding: The Algorithm

- Given D dim. data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, compute K dim. projections $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$
- For each example $\mathbf{x}_i$, find its L nearest neighbors
- Assume $\mathbf{x}_i$ to be a weighted linear combination of the L nearest neighbors

$$\mathbf{x}_i \approx \sum_{j \in \mathcal{N}} W_{ij} \mathbf{x}_j \qquad \text{(so the data is assumed locally linear)}$$

- Find the weights by solving the following least-squares problem:

$$W = \arg\min_W \sum_{i=1}^{N} ||\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} W_{ij} \mathbf{x}_j||^2 \qquad s.t. \forall i \quad \sum_j W_{ij} = 1$$

- $\mathcal{N}_i$ are the L nearest neighbors of $\mathbf{x}_i$ (note: should choose $L \geq K + 1$)
- Use W to compute low dim. projections $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ by solving:

$$\mathbf{Z} = \arg\min_{\mathbf{Z}} \sum_{i=1}^{N} ||\mathbf{z}_i - \sum_{j \in \mathcal{N}} W_{ij} \mathbf{z}_j||^2 \qquad s.t. \forall i \quad \sum_{i=1}^{N} \mathbf{z}_i = 0, \quad \frac{1}{N} \mathbf{Z}\mathbf{Z}^\top = \mathbf{I}$$

Locally Linear Embedding: The Algorithm

- Given D dim. data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, compute K dim. projections $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$
- For each example $\mathbf{x}_i$, find its L nearest neighbors
- Assume $\mathbf{x}_i$ to be a weighted linear combination of the L nearest neighbors

$$\mathbf{x}_i \approx \sum_{j \in \mathcal{N}} W_{ij}\mathbf{x}_j \qquad \text{(so the data is assumed locally linear)}$$

- Find the weights by solving the following least-squares problem:

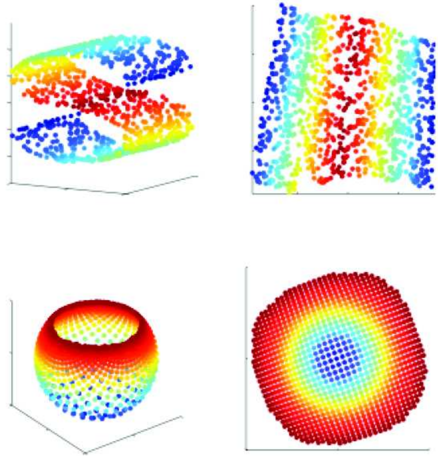
$$W = \arg\min_{W} \sum_{i=1}^{N} ||\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} W_{ij}\mathbf{x}_j||^2 \qquad s.t. \forall i \quad \sum_j W_{ij} = 1$$

- $\mathcal{N}_i$ are the L nearest neighbors of $\mathbf{x}_i$ (note: should choose $L \geq K + 1$)
- Use W to compute low dim. projections $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ by solving:

$$\mathbf{Z} = \arg\min_{\mathbf{Z}} \sum_{i=1}^{N} ||\mathbf{z}_i - \sum_{j \in \mathcal{N}} W_{ij}\mathbf{z}_j||^2 \qquad s.t. \forall i \quad \sum_{i=1}^{N} \mathbf{z}_i = 0, \quad \frac{1}{N}\mathbf{Z}\mathbf{Z}^\top = \mathbf{I}$$

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

LLE: Examples



Isometric Feature Mapping (Isomap)

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

Isometric Feature Mapping (Isomap)

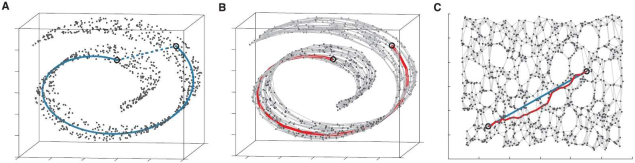
A graph based algorithm based on constructing a matrix of geodesic distances

- 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)

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 $\mathbf{D} = \{d_{ij}^2\}$



Isomap (Contd.)

- Use the distance matrix $\mathbf{D}$ to construct the Gram Matrix

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

where $\mathbf{G}$ is $N \times N$ and

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

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

Isomap (Contd.)

- Use the distance matrix $\mathbf{D}$ to construct the Gram Matrix

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

where $\mathbf{G}$ is $N \times N$ and

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

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

- Do an eigen decomposition of $\mathbf{G}$
- Let the eigenvectors be $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ with eigenvalues $\{\lambda_1, \ldots, \lambda_N\}$
  - Each eigenvector $\mathbf{v}_i$ is N-dimensional: $\mathbf{v}_i = [v_{1i}, v_{2i}, \ldots, v_{Ni}]$
- Take the top K eigenvalue/eigenvectors

Isomap (Contd.)

- Use the distance matrix $\mathbf{D}$ to construct the Gram Matrix

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

where $\mathbf{G}$ is $N \times N$ and

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

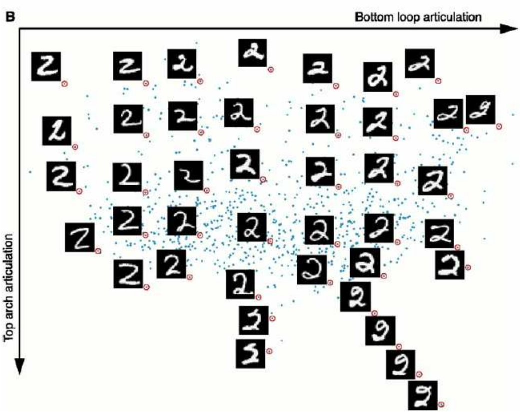
$\mathbf{I}$ is $N \times N$ identity matrix, $\mathbf{1}$ is $N \times 1$ vector of 1s

- Do an eigen decomposition of $\mathbf{G}$
- Let the eigenvectors be $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ with eigenvalues $\{\lambda_1, \ldots, \lambda_N\}$
    - Each eigenvector $\mathbf{v}_i$ is N-dimensional: $\mathbf{v}_i = [v_{1i}, v_{2i}, \ldots, v_{Ni}]$
- Take the top K eigenvalue/eigenvectors
- The K dimensional embedding $\mathbf{z}_i = [z_{i1}, z_{i2}, \ldots, z_{iK}]$ of a point $\mathbf{x}_i$:

$$z_{ik} = \sqrt{\lambda_k} v_{ki}$$

Isomap: Example

Digit images projected down to 2 dimensions



Isomap: Example

Face images with varying poses