Visualization
( Nonlinear dimensionality reduction )

Fei Sha

Yahoo! Research
feisha@yahoo-inc.com

CS294  March 18, 2008
Question:
How can we detect *low dimensional structure* in *high dimensional* data?

Motivations:
- Exploratory data analysis & visualization
- Compact representation
- Robust statistical modeling
Linear dimensionality reductions

- Many examples (Percy’s lecture on 2/19/2008)
  - Principal component analysis (PCA)
  - Fischer discriminant analysis (FDA)
  - Nonnegative matrix factorization (NMF)

- Framework

  \[ x \in \mathbb{R}^D \rightarrow y \in \mathbb{R}^d \]

  \[ D \gg d \]

  \[ y = Ux \]

  linear transformation of original space
Linear methods are not sufficient

- What if data is “nonlinear”?

classic toy example of Swiss roll

- PCA results
What we really want is “unrolling”

Simple geometric intuition:

distortion in local areas
faithful in global structure
Outline

- **Linear method: redux and new intuition**
  Multidimensional scaling (MDS)

- **Graph based spectral methods**
  - Isomap
  - Locally linear embedding

- **Other nonlinear methods**
  - Kernel PCA
  - Maximum variance unfolding (MVU)
Linear methods: redux

PCA: does the data mostly lie in a subspace? If so, what is its dimensionality?

\begin{itemize}
  \item \(D = 2\) \quad \(d = 1\)
  \item \(D = 3\) \quad \(d = 2\)
\end{itemize}
The framework of PCA

• **Assumption:**
  
  Centered inputs
  
  Projection into subspace

• **Interpretation**
  
  maximum variance preservation
  
  minimum reconstruction errors

\[
\sum_i x_i = 0
\]

\[
y_i = U x_i
\]

\[
UU^T = I
\]

(note: a small change from Percy’s notation)
Other criteria we can think of...

How about preserve pairwise distances?

\[ \| \mathbf{x}_i - \mathbf{x}_j \| = \| \mathbf{y}_i - \mathbf{y}_j \| \]

This leads to a new type of linear methods:

multidimensional scaling (MDS)

Key observation: from distances to inner products

\[ \| \mathbf{x}_i - \mathbf{x}_j \|^2 = \mathbf{x}_i^T \mathbf{x}_i - 2 \mathbf{x}_i^T \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j \]
Recipe for multidimensional scaling

- Compute Gram matrix on centered points
  \[ G = X^T X \]
  \[ X = (x_1, x_2, \ldots, x_N) \]
- Diagonalize
  \[ G = \sum_i \lambda_i v_i v_i^T \]
  \[ \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \]
- Derive outputs and estimate dimensionality
  \[ d = \min \arg \max 1 \left( \sum_{i=1}^{\text{d}} \lambda_i \geq \text{THRESHOLD} \right) \]
  \[ y_{id} = \sqrt{\lambda_i v_{id}} \]
MDS when only distances are known

We convert distance matrix

\[ D = \{ d_{ij}^2 \} \]

\[ d_{ij}^2 = \| x_i - x_j \|^2 \]

to Gram matrix

\[ G = -\frac{1}{2} HDH \]

with centering matrix

\[ H = I_n - \frac{1}{n} 11^T \]
PCA vs MDS: is MDS really that new?

- Same set of eigenvalues

\[
\frac{1}{N}XX^T \mathbf{v} = \lambda \mathbf{v} \quad \rightarrow \quad X^T X \frac{1}{N}X^T \mathbf{v} = N\lambda \frac{1}{N}X^T \mathbf{v}
\]

- Similar low dimensional representation

- Different computational cost

PCA scales quadratically in D
MDS scales quadratically in N

Big win for MDS when D is much greater than N!
How to generalize to nonlinear structures?

All we need is a simple twist on MDS
5min Break?
Nonlinear structures

- Manifolds such as

- can be approximately locally with linear structures.

This is a key intuition that we will repeatedly appeal to
Manifold learning

Given high dimensional data sampled from a low dimensional nonlinear submanifold, how to compute a faithful embedding?

Input
\[ \{ x_i \in \mathbb{R}^D, i = 1, 2, \ldots, n \} \]

Output
\[ \{ y_i \in \mathbb{R}^d, i = 1, 2, \ldots, n \} \]
Outline

- **Linear method: redux and new intuition**
  Multidimensional scaling (MDS)

- **Graph based spectral methods**
  Isomap

  Locally linear embedding

- **Other nonlinear methods**
  Kernel PCA
  Maximum variance unfolding
A small jump from MDS to Isomap

- Key idea
  - MDS
  - Preserve pairwise Euclidean distances
A small jump from MDS to Isomap

- **Key idea**
  
  Isomap

  Preserve pairwise geodesic distances

- **Algorithm in a nutshell**

  Estimate geodesic distance along submanifold

  Perform MDS as if the distances are Euclidean
Why geodesic distances?

Euclidean distance is not appropriate measure of proximity between points on nonlinear manifold.

A closer to C in Euclidean distance

A closer to B in geodesic distance
Caveat

Without knowing the shape of the manifold, how to estimate the geodesic distance?

The tricks will unfold next....
Step 1. Build adjacency graph

- **Graph from nearest neighbor**
  
  Vertices represent inputs
  
  Edges connect nearest neighbors

- **How to choose nearest neighbor**
  
  k-nearest neighbors
  
  Epsilon-radius ball

**Q: Why nearest neighbors?**

**A1: local information more reliable than global information**

**A2: geodesic distance \( \approx \) Euclidean distance**
Building the graph

- **Computation cost**
  
  kNN scales naively as $O(N^2D)$
  
  Faster methods exploit data structure (e.g., KD-tree)

- **Assumptions**
  
  Graph is connected (if not, run algorithms on each connected component)
  
  No short-circuit

  Large $k$ would cause this problem
Step 2. Construct geodesic distance matrix

- Geodesic distances
  Weight edges by local Euclidean distance
  Approximate geodesic by shortest paths

- Computational cost
  Require all pair shortest paths (Dijkstra’s algorithm: $O(N^2 \log N + N^2k)$)
  Require dense sampling to approximate well
  (very intensive for large graph)
Step 3. Apply MDS

- Convert geodesic matrix to Gram matrix
  Pretend the geodesic matrix is from Euclidean distance matrix
- Diagonalize the Gram matrix
  Gram matrix is a dense matrix, ie, no sparsity
  Can be intensive if the graph is big.
- Embedding
  Number of significant eigenvalues yield estimate of dimensionality
  Top eigenvectors yield embedding.
Quick summary

• Build nearest neighbor graph

• Estimate geodesic distances

• Apply MDS

This would be a recurring theme for many graph based manifold learning algorithms.
Examples

- Swiss roll
  - $N = 1024$
  - $k = 12$

- Digit images
  - $N = 1000$
  - $r = 4.2$
  - $D = 400$
Applications: Isomap for music

Embedding of sparse music similarity graph (Platt, NIPS 2004)

\[N = 267,000\]
\[E = 3.22 \text{ million}\]
Outline

- Linear method: redux and new intuition
  Multidimensional scaling (MDS)
- Graph based spectral methods
  Isomap
- Locally linear embedding
- Other nonlinear methods
  Kernel PCA
  Maximum variance unfolding
Locally linear embedding (LLE)

• **Intuition**

Better off being myopic and trusting only local information

• **Steps**

Define locality by nearest neighbors
Encode local information** Least square fit locally**
Minimize global objective to preserve local information** Think globally**
Step 1. Build adjacency graph

- **Graph from nearest neighbor**
  Vertices represent inputs
  Edges connect nearest neighbors

- **How to choose nearest neighbor**
  k-nearest neighbors
  Epsilon-radius ball

This step is exactly the same as in Isomap.
Step 2. Least square fits

- Characterize local geometry of each neighborhood by a set of weights

\[ \Phi(W) = \sum_i \| x_i - \sum_k W_{ik} x_k \|^2 \]

subject to \[ \sum_k W_{ik} = 1 \]

- Compute weights by reconstructing each input linearly from its neighbors
What are these weights for?

They are shift, rotation, scale invariant.

The head should sit in the middle of left and right finger tips.
Step 3. Preserve local information

- The embedding should follow the same local encoding

\[ y_i \approx \sum_k W_{ik} y_k \]

- Minimize a \textbf{global} reconstruction error

\[ \Psi(Y) = \sum_i \| y_i - \sum_k W_{ik} y_k \|^2 \]

subject to

\[ \sum y_i = 0 \]

\[ \frac{1}{N} YY^T = I \]
Sparse eigenvalue problem

- Quadratic form

$$\arg\min \Psi(Y) = \sum_{i,j} \Psi_{ij} y_i^T y_j$$

$$\Psi = (I - W)^T (I - W)$$

- Rayleigh-Ritz quotient

Embedding given by bottom eigenvectors

Discard bottom eigenvector $[1 1 \ldots 1]$

Other $d$ eigenvectors yield embedding
Summary

- Build k-nearest neighbor graph
- Solve linear least square fit for each neighbor
- Solve a sparse eigenvalue problem

Every step is relatively trivial, however the combined effect is quite complicated.
Examples

\( N = 1000 \)
\( k = 8 \)
\( D = 3 \)
\( d = 2 \)
Examples of LLE

- Pose and expression

N = 1965
k = 12
D = 560
d = 2
## Recap: Isomap vs. LLE

<table>
<thead>
<tr>
<th></th>
<th>Isomap</th>
<th>LLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve</td>
<td><strong>geodesic distance</strong></td>
<td><strong>local symmetry</strong></td>
</tr>
<tr>
<td>construct nearest</td>
<td><strong>nearest neighbor graph; formulate quadratic form; diagonalize</strong></td>
<td><strong>nearest neighbor graph; formulate quadratic form; diagonalize</strong></td>
</tr>
<tr>
<td></td>
<td>pick <strong>top eigenvector</strong>; estimate dimensionality</td>
<td>pick <strong>bottom eigenvector; does not estimate dimensionality</strong></td>
</tr>
<tr>
<td></td>
<td><strong>more computationally expensive</strong></td>
<td><strong>much more contractable</strong></td>
</tr>
</tbody>
</table>
There are still many

- Laplacian eigenmaps
- Hessian LLE
- Local Tangent Space Analysis
- Maximum variance unfolding
- ...

...
Summary: graph based spectral methods

• **Construct nearest neighbor graph**
  Vertices are data points
  Edges indicate nearest neighbors

• **Spectral decomposition**
  Formulate matrix from the graph
  Diagonalize the matrix

• **Derive embedding**
  Eigenvector as embedding
  Estimate dimensionality
5min Break?
Outline

- Linear method: redux and new intuition
  Multidimensional scaling (MDS)
- Graph based spectral methods
  Isomap
  Locally linear embedding
- Other nonlinear methods
  Kernel PCA
  Maximum variance unfolding
Another twist on MDS to get nonlinearity

- **Key idea**

  Map data points with nonlinear functions

  \[ \phi : \mathbf{x} \rightarrow \phi(\mathbf{x}) \]

  Perform PCA/MDS in the new space

  \[ \phi(\mathbf{X})^T \phi(\mathbf{X}) \mathbf{v} = \lambda \mathbf{v} \]

  *(MDS: diagonalizing Gram matrix)*
The kernel trick

The inner product

$$\phi(x_i)^T \phi(x_j)$$

is more interesting than the exact form of the mapping function.

For certain mapping function, we can find a kernel function

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Therefore, all we need to do is to specify a kernel function to find the projections!
Kernel PCA

• Algorithm

Select a kernel: Gaussian kernel, string kernel

Construct kernel matrix \( K = [K_{ij}] = [K(x_i, x_j)] \)

Diagonalize the kernel matrix

• Caveat

Kernel PCA does not always reduce dimensions.
Very important in choosing appropriate kernel
Heavy computation for large data sets
Why would we want to use kernels?

- **Handle complex data types.**

  Kernels for numerical data (e.g., CPU load)
  \[
  K(x_i, x_j) = \exp\left(-\|x_i - x_j\|^2 / \sigma\right)
  \]

  “String” kernels for text data (e.g., URL/HTTP request)
  \[
  K(s_i, s_j) = \# of common substrings
  \]

- **Building blocks**

  Multiple kernels can be combined into a single kernel
Outline

• **Linear method: redux and new intuition**
  Multidimensional scaling (MDS)

• **Graph based spectral methods**
  Isomap
  Locally linear embedding

• **Other nonlinear methods**
  Kernel PCA

**Maximum variance unfolding**
Enforcing distance constraints explicitly

- **Quadratic programming**
  \[
  \max \sum_i \|y_i\|^2 \\
  \sum_i y_i = 0 \\
  \|y_i - y_j\|^2 = \|x_i - x_j\|^2
  \]
  unfolding centering only if i and j are nearest neighbor!

- **Intuition**
  Nearby points are connected with rigid rods
  Unfold inputs without breaking apart rods.

  Rotation allowed
Convex optimization

- Change of variables

\[ K_{ij} = y_i^T y_j \]

- Semidefinite programming (SDP)

\[
\begin{align*}
\text{max} & \quad \sum_{ii} K_{ii} \\
\sum_{ij} K_{ij} & = 0 \\
K_{ii} + K_{jj} - 2K_{ij} & = \|x_i - x_j\|^2 \\
K & \succeq 0
\end{align*}
\]

See this trick before?

Gram matrix needs to be positive semidefinite
Outline of the MVU algorithm

• Compute nearest neighbors & local distances
• Solve SDP
  Convex optimization
  Use off-shelf SDP solver
• Analyze the SDP solution
  Apply MDS to the kernel matrix
  Yield embedding and dimensionality

Implementation: complicated and non-trivial; best bet to use others’ package
Images of rotating teapot

- Full rotation
  
  \[N = 400\]
  \[k = 4\]
  \[D = 23028\]

- Half rotation

Images are ordered by d=1 embedding according to view angle
MVU vs. Isomap

- **Similarities**
  - Both motivated by isometry
  - Based on constructing Gram matrix
  - Eigenvalues reveal dimensionality

- **Differences**
  - Semidefinite vs. dynamic programing to find Gram matrix
  - Finite vs. asymptotic guarantee
  - MVU works for manifolds with “holes”
Application: sensor localization

Infer coordinates from limited measurement of distances

(Weinberger, Sha & Saul, NIPS 2006)
Embedding in 2D while ignoring distances

Turn distance matrix into adjacency matrix
Compute 2D embedding with Laplacian eigenmaps
Assumption: measurements exist only if sensors are close to each other
Adding distance constraints

Start from Laplacian eigenmap results
Enforce known distances constraints
Find embedding using maximum variance unfolding
Recover almost perfectly!
Conclusion

• **Big picture**

  Large-scale high dimensional data everywhere.
  Many of them have intrinsic low dimension representation.
  Nonlinear techniques can be very helpful for exploratory data analysis and visualization.

• **Techniques we sampled today**

  Manifold learning techniques.
  Kernel methods.
Resources

- Manifold learning tutorials by Lawrence K. Saul (UCSD)
  
  http://www.cs.ucsd.edu/~saul/tutorials.html

- A bookmark page for manifold learning
  
  http://www.cse.msu.edu/~lawhiu/manifold/
Software

• Matlab learning demo

http://www.math.umn.edu/~wittman/mani/

• Manifold learning toolbox

http://www.cs.unimaas.nl/l.vandermaaten/Laurens_van_der_Maaten/ Matlab_Toolbox_for_Dimensionality_Reduction.html