Visualization
(Nonlinear dimensionality reduction)

Fei Sha

Yahoo! Research
feisha@yahoo-inc.com

CS294 March 18, 2008

Dimensionality reduction

- **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)

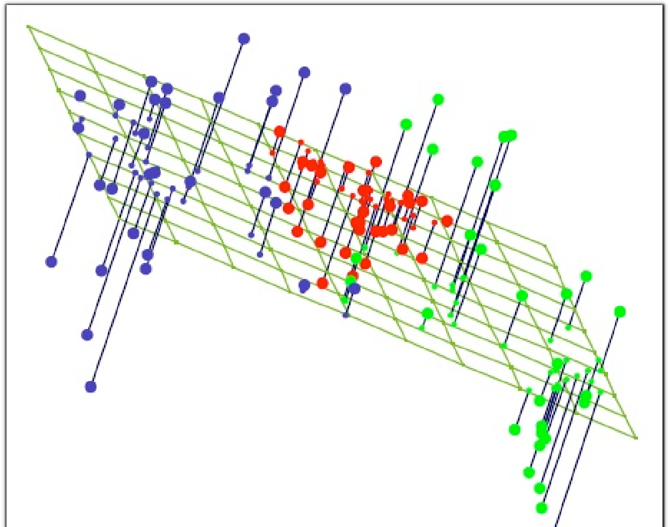
  Nonnegative matrix factorization (NMF)

- **Framework**

$$\boldsymbol{x} \in \Re^D \rightarrow \boldsymbol{y} \in \Re^d$$

$$D \gg d$$

$$\boldsymbol{y} = \boldsymbol{U}\boldsymbol{x}$$

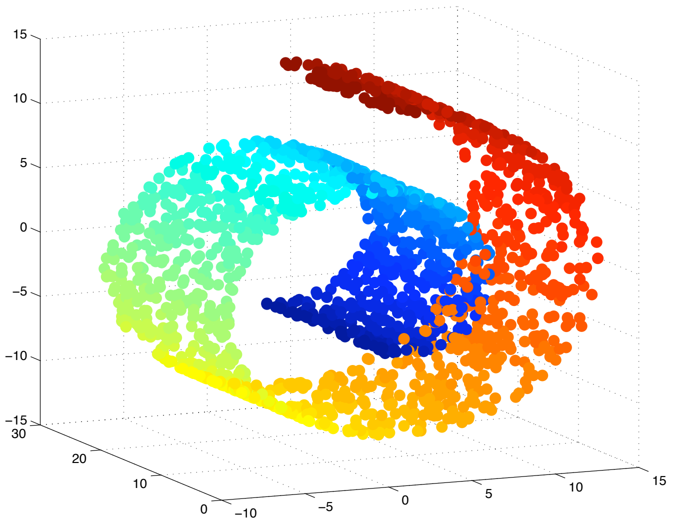


linear transformation of original space

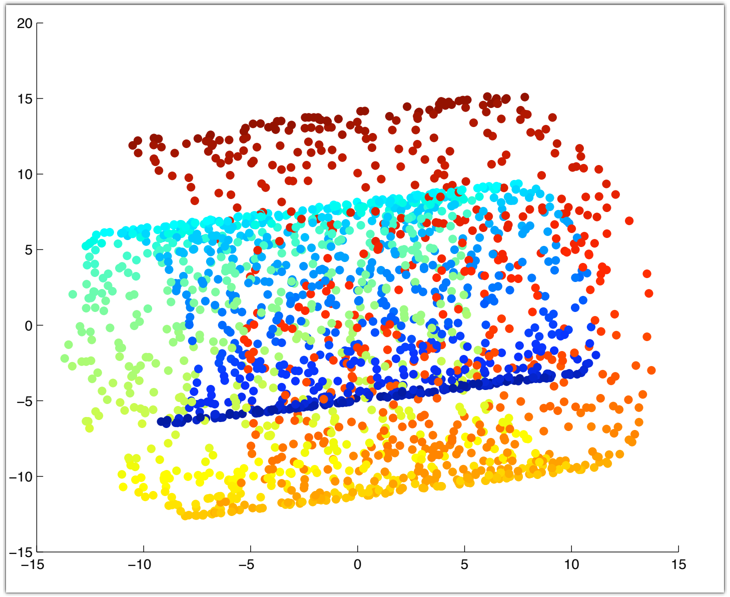
Linear methods are not sufficient

- **What if data is "nonlinear"?**

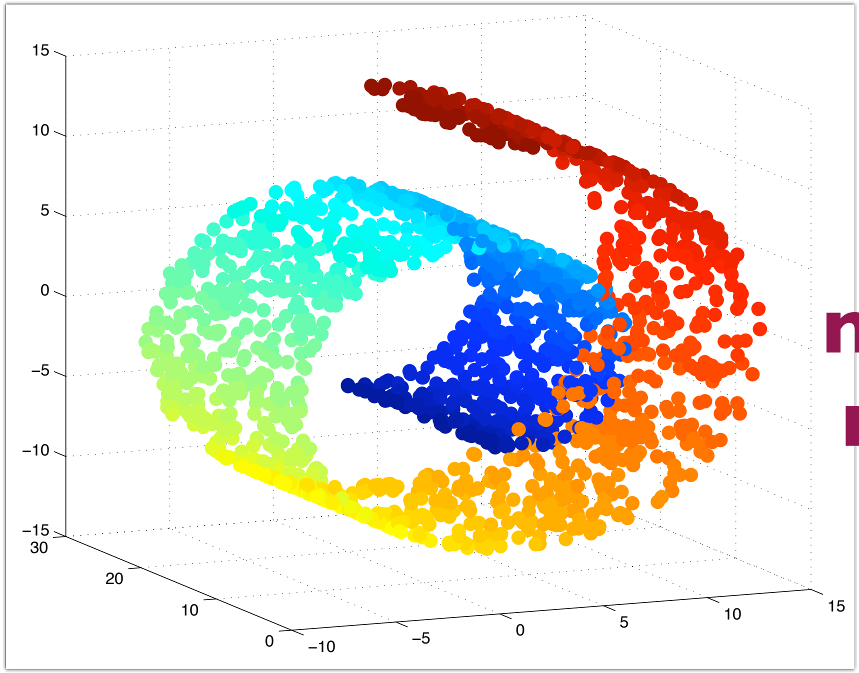
classic toy example of Swiss roll



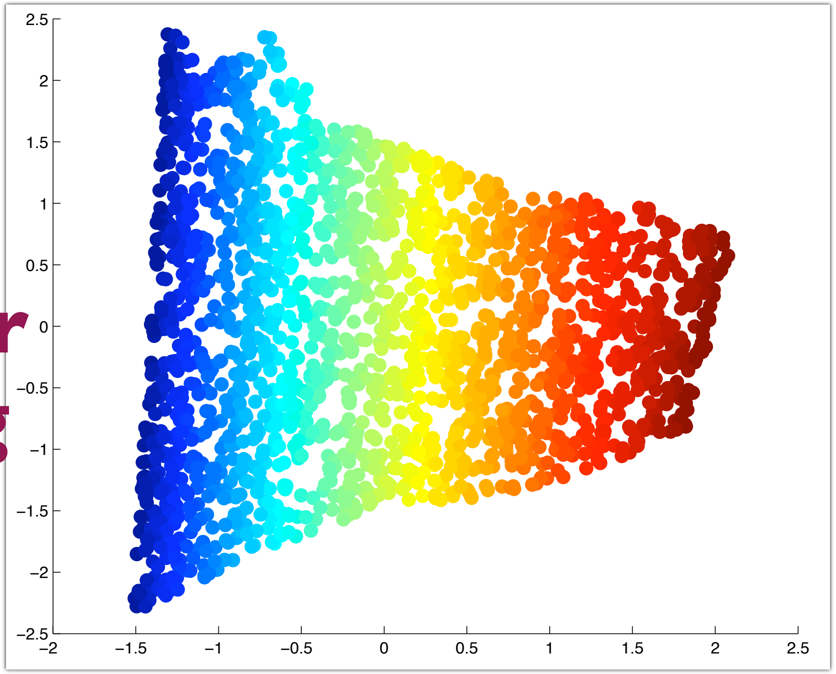
- **PCA results**



What we really want is “unrolling”



nonlinear mapping



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

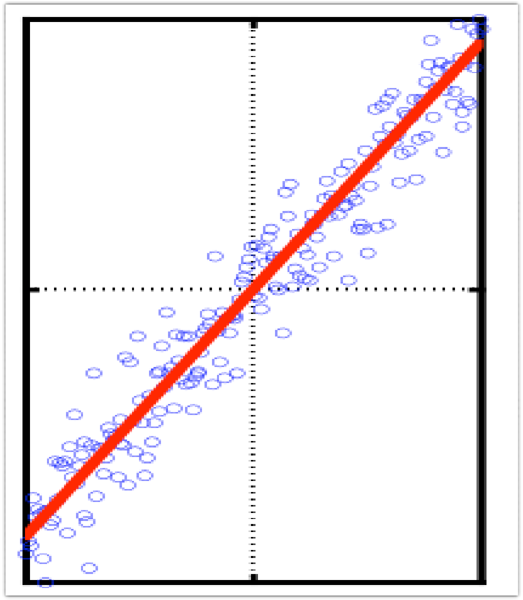
  Kernel PCA

  Maximum variance unfolding (MVU)

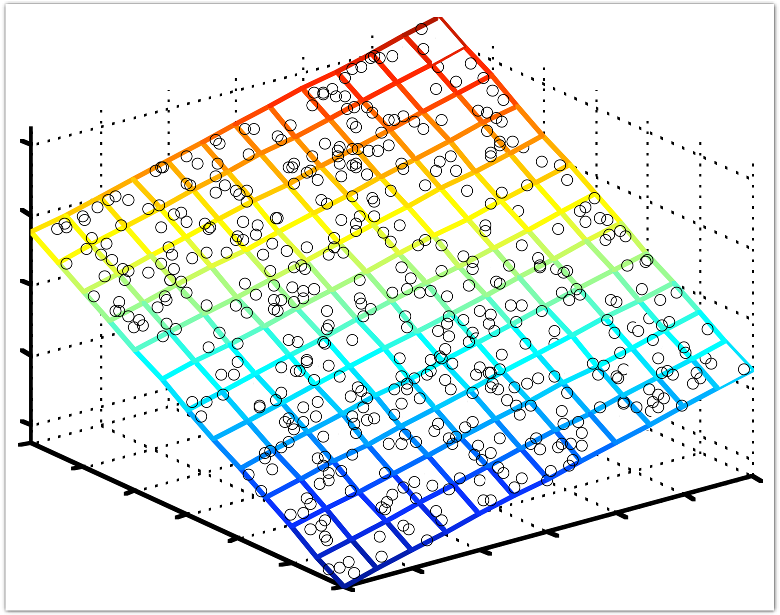
Linear methods: redux

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

D = 2
d = 1



D = 3
d = 2



The framework of PCA

- **Assumption:**

  Centered inputs $$\sum_i \boldsymbol{x}_i = \boldsymbol{0}$$

  Projection into subspace $$\boldsymbol{y}_i = \boldsymbol{U}\boldsymbol{x}_i$$

  $$\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}} = \boldsymbol{I}$$

  (note: a small change from Percy's notation)

- **Interpretation**

  maximum variance preservation

  $$\arg\max \sum_i \|\boldsymbol{y}_i\|^2$$

  minimum reconstruction errors

  $$\arg\min \sum_i \|\boldsymbol{x}_i - \boldsymbol{U}^{\mathrm{T}}\boldsymbol{y}_i\|^2$$

Other criteria we can think of...

How about **preserve pairwise distances**?

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = \|\boldsymbol{y}_i - \boldsymbol{y}_j\|$$

This leads to a new type of linear methods

multidimensional scaling (MDS)

Key observation: from distances to inner products

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i - 2\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{x}_j$$

Recipe for multidimensional scaling

- **Compute Gram matrix on centered points**

$$\boldsymbol{G} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$$

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$$

- **Diagonalize**

$$\boldsymbol{G} = \sum_i \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^{\mathrm{T}} \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$$

- **Derive outputs and estimate dimensionality**

$$d = \min \arg\max \mathbb{1}\left(\sum_{i=1}^{d} \lambda_i \geq \mathrm{THRESHOLD}\right)$$

$$y_{id} = \sqrt{\lambda_i} v_{id}$$

MDS when only distances are known

We convert distance matrix

$$\boldsymbol{D} = \{d_{ij}^2\} \qquad d_{ij}^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$$

to Gram matrix

$$\boldsymbol{G} = -\frac{1}{2}\boldsymbol{H}\boldsymbol{D}\boldsymbol{H}$$

with centering matrix

$$\boldsymbol{H} = \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$$

PCA vs MDS: is MDS really that new?

- Same set of eigenvalues

$$\frac{1}{N}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}v = \lambda v \rightarrow \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\frac{1}{N}\boldsymbol{X}^{\mathrm{T}}v = N\lambda\frac{1}{N}\boldsymbol{X}^{\mathrm{T}}v$$

PCA diagonalization **MDS diagonalization**

- Similar low dimensional representation
- Different computational cost

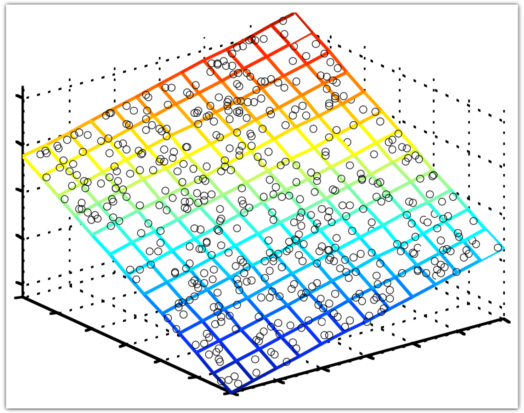
PCA scales quadratically in D

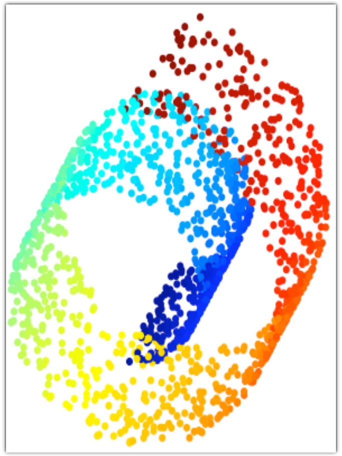
MDS scales quadratically in N

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

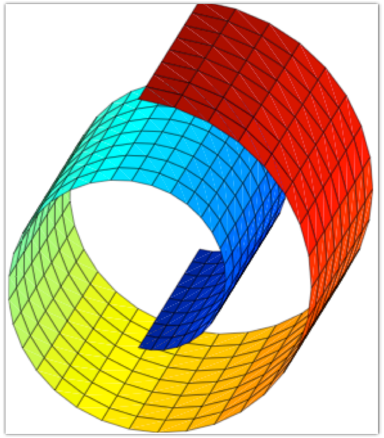


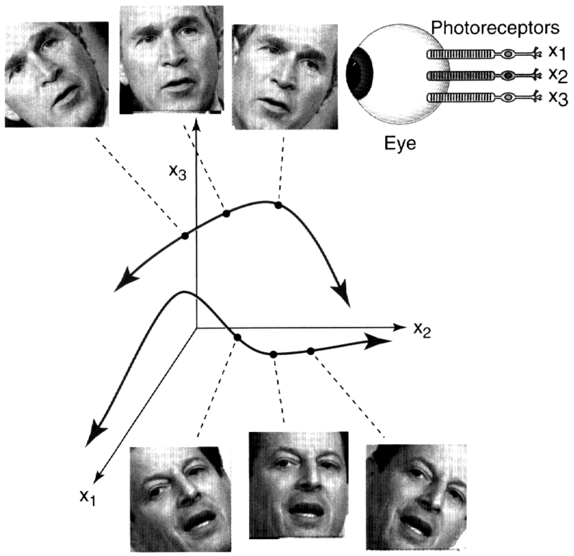


5min Break?

Nonlinear structures

- Manifolds such as



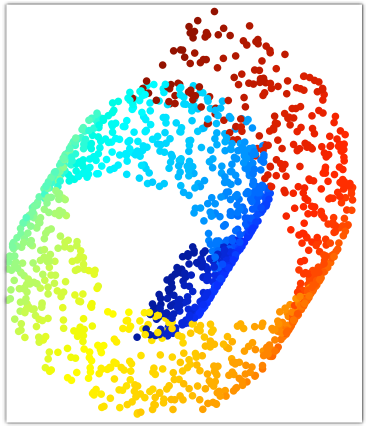


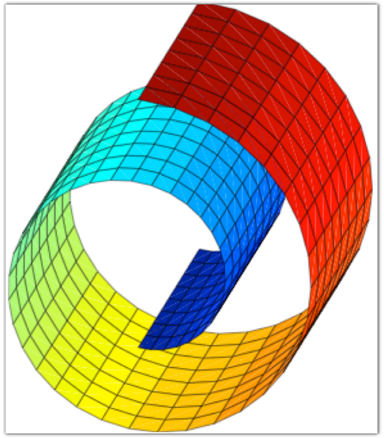
- can be approximately locally with linear structures.

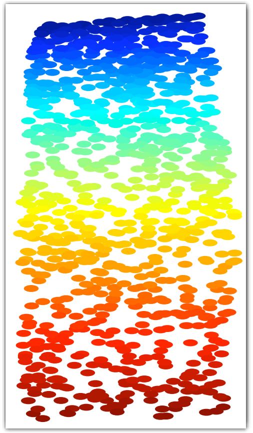
This is a key intuition that we will repeatedly appeal to

Manifold learning

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







Input

$$\{\boldsymbol{x}_i \in \Re^D, i = 1, 2, \ldots, n\}$$

Output

$$\{\boldsymbol{y}_i \in \Re^d, i = 1, 2, \ldots, n\}$$

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

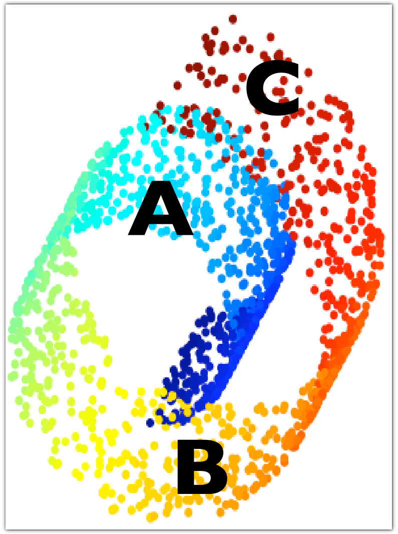
- **Algorithm in a nutshell**

  Estimate geodesic distance along submanifold

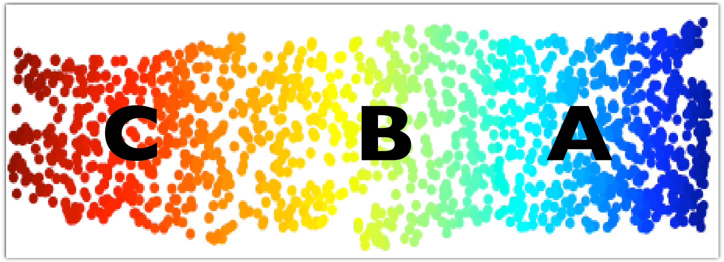
  Perform MDS as if the distances are Euclidean

Why geodesic distances?

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



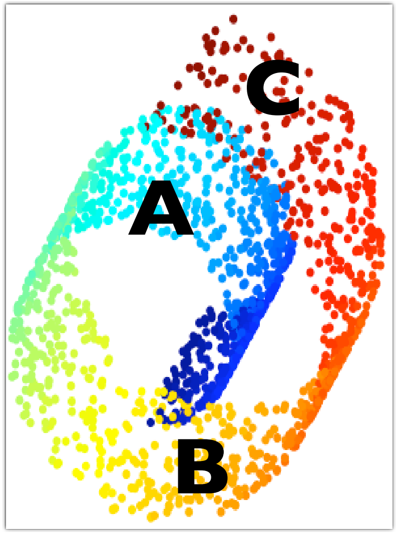
A closer to C in Euclidean distance



A closer to B in geodesic distance

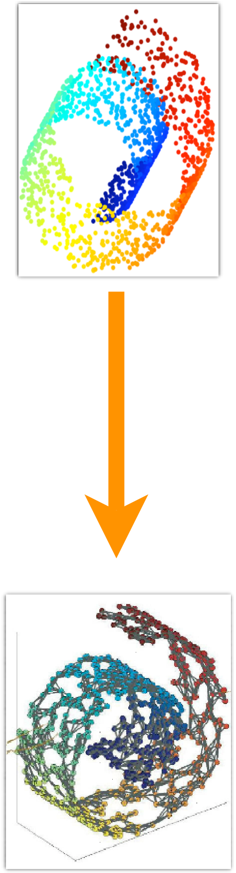
Caveat

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 $O(N^2D)$

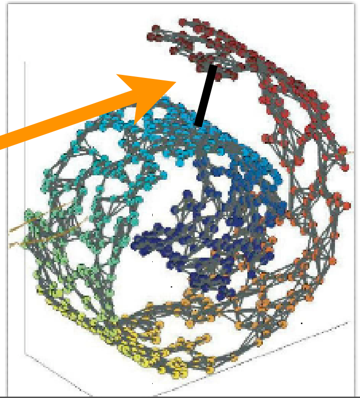
  Faster methods exploit data structure (eg, KD-tree)

- **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 (Djikstra's algorithm: $O(N^2 \log N + N^2 k)$)

  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.

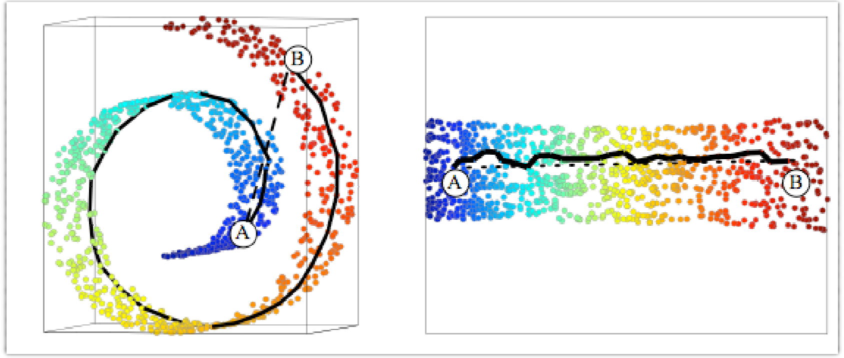
Quick summary

- Build nearest neighbor graph
- Estimate geodesic distances
- Apply MDS

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

Examples

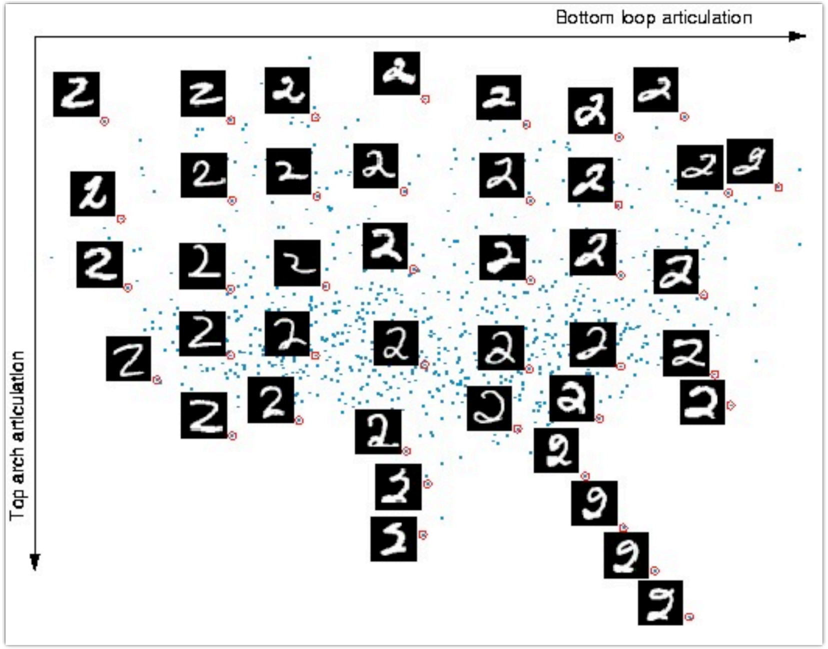
- **Swiss roll**



N = 1024
k = 12

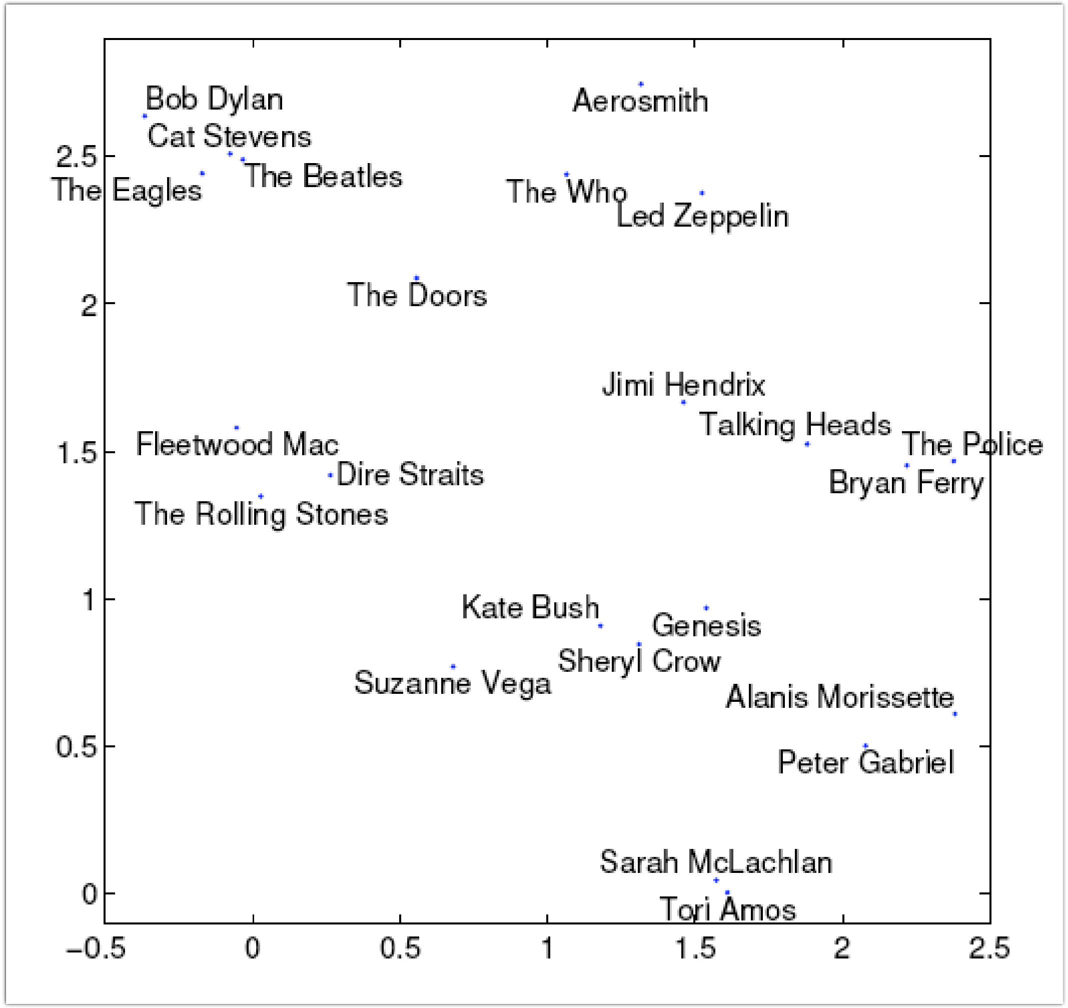
- **Digit images**

  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 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**

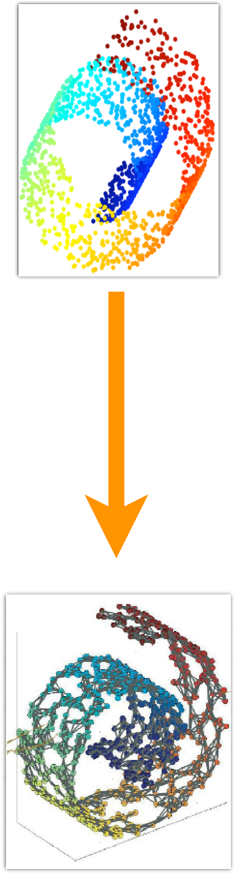
  Vertices represent inputs

  Edges connect nearest neighbors

- **How to choose nearest neighbor**

  k-nearest neighbors

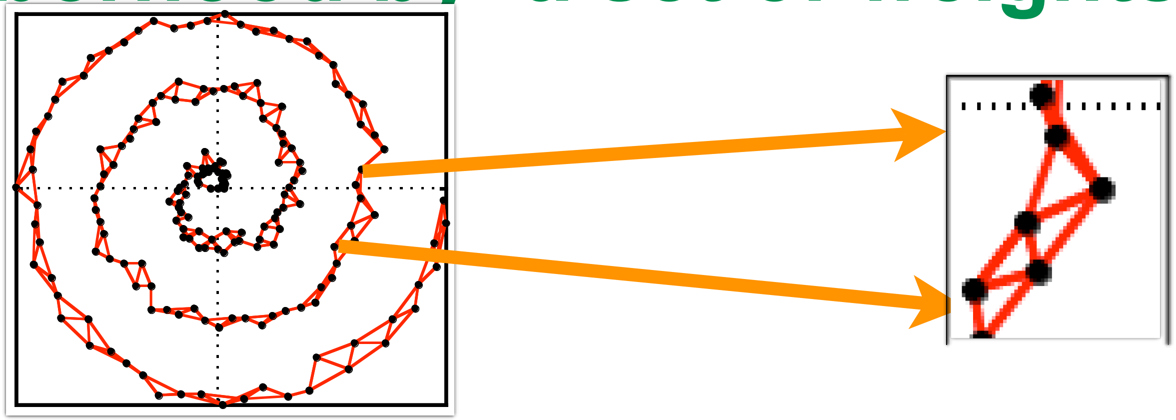
  Epsilon-radius ball



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



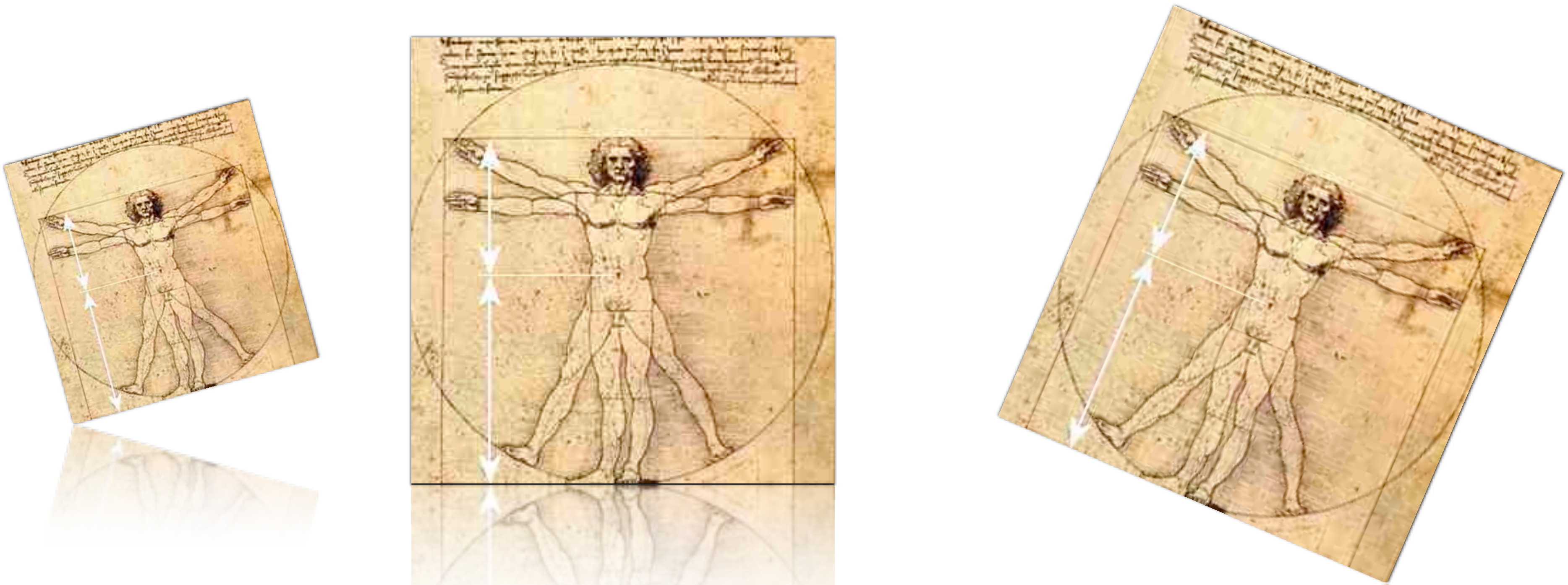
- Compute weights by reconstructing each input linearly from its neighbors

$$\Phi(\boldsymbol{W}) = \sum_i \|\boldsymbol{x}_i - \sum_k \boldsymbol{W}_{ik}\boldsymbol{x}_k\|^2$$

subject to $\sum_k W_{ik} = 1$

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 same local encoding

$$\boldsymbol{y}_i \approx \sum_k \boldsymbol{W}_{ik} \boldsymbol{y}_k$$

- Minimize a global reconstruction error

$$\Psi(\boldsymbol{Y}) = \sum_i \|\boldsymbol{y}_i - \sum_k \boldsymbol{W}_{ik} \boldsymbol{y}_k\|^2$$

subject to

$$\sum \boldsymbol{y}_i = \boldsymbol{0}$$

$$\frac{1}{N} \boldsymbol{Y} \boldsymbol{Y}^{\mathrm{T}} = \boldsymbol{I}$$

Sparse eigenvalue problem

- **Quadratic form**

$$\arg\min \Psi(\boldsymbol{Y}) = \sum_{ij} \Psi_{ij} \boldsymbol{y}_i^{\mathrm{T}} \boldsymbol{y}_j$$

$$\boldsymbol{\Psi} = (\boldsymbol{I} - \boldsymbol{W})^{\mathrm{T}} (\boldsymbol{I} - \boldsymbol{W})$$

- **Rayleigh-Ritz quotient**

Embedding given by bottom eigenvectors

Discard bottom eigenvector [1 1 ... 1]

Other d eigenvectors yield embedding

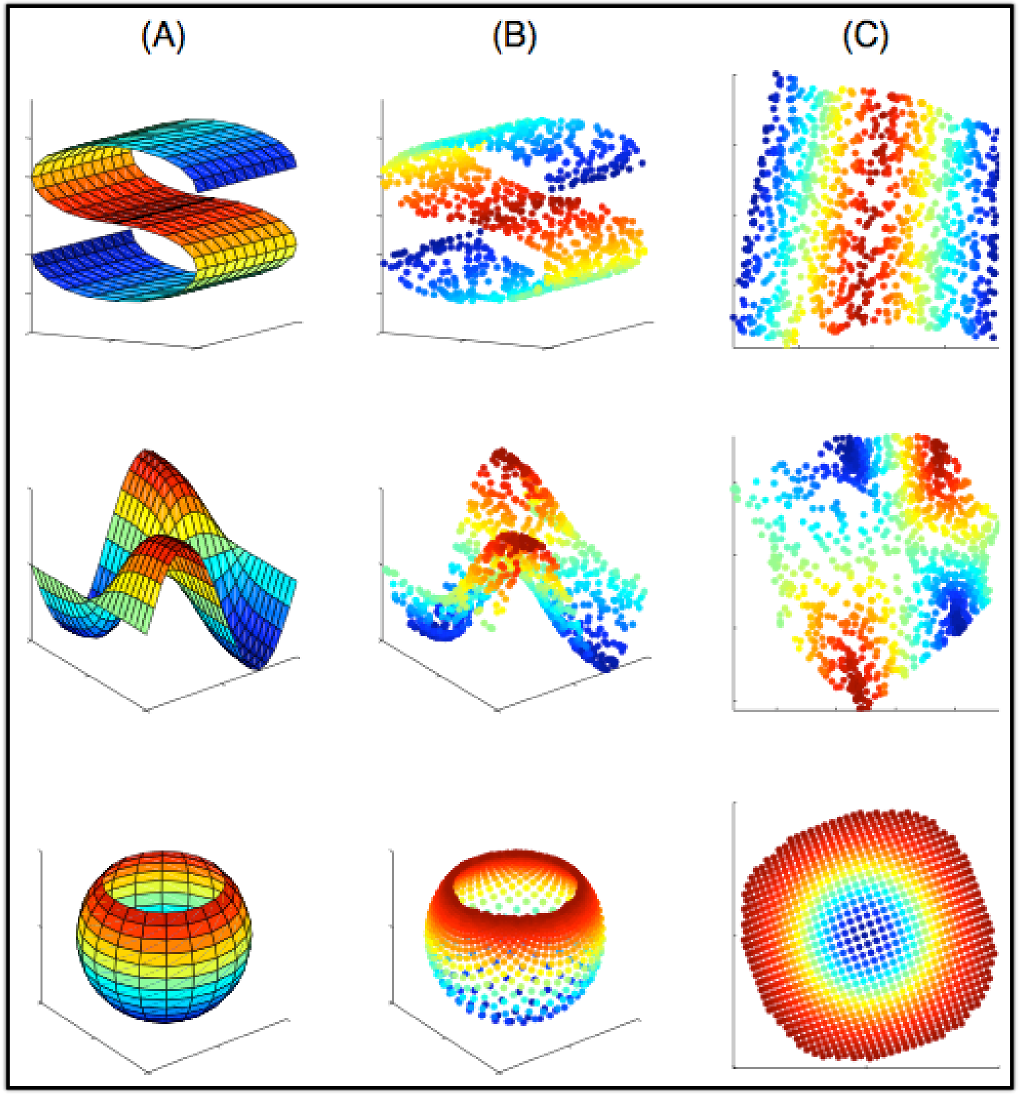
Summary

- 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

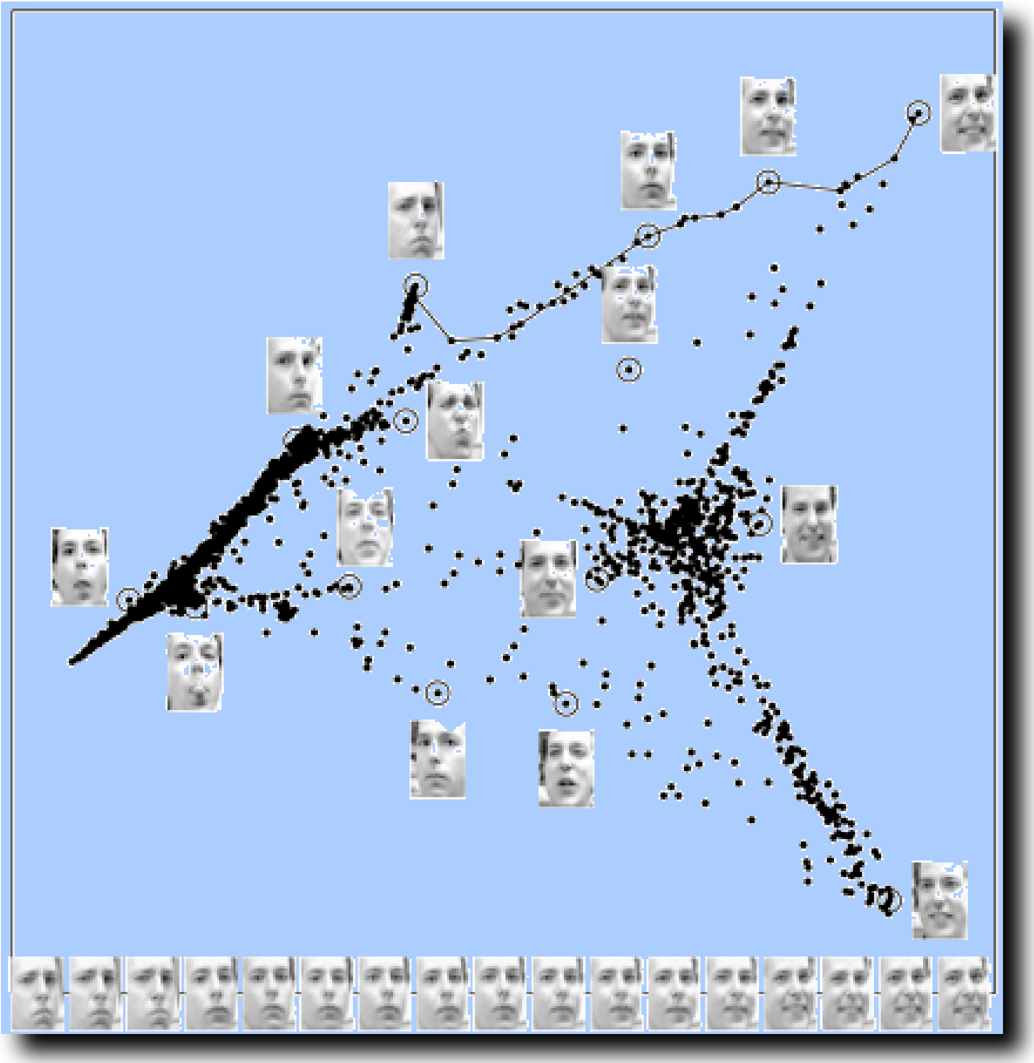
N = 1000
k = 8
D = 3
d = 2



Examples of LLE

- Pose and expression

N = 1965
k = 12
D = 560
d = 2



Recap: Isomap vs. LLE

Isomap	LLE
Preserve geodesic distance	Preserve local symmetry
construct nearest neighbor graph; formulate quadratic form; diagonalize	construct nearest neighbor graph; formulate quadratic form; diagonalize
pick top eigenvector; estimate dimensionality	pick bottom eigenvector; does not estimate dimensionality
more computationally expensive	much more contractable

There are still many

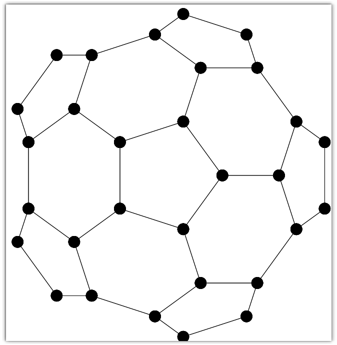
- Laplacian eigenmaps
- Hessian LLE
- Local Tangent Space Analysis
- Maximum variance unfolding
- ...

Summary: graph based spectral methods

- **Construct nearest neighbor graph**

  Vertices are data points

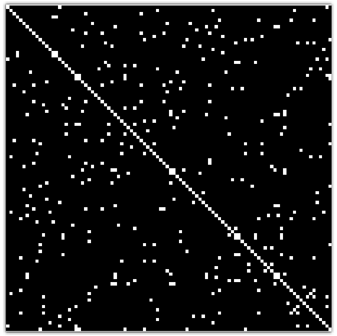
  Edges indicate nearest neighbors



- **Spectral decomposition**

  Formulate matrix from the graph

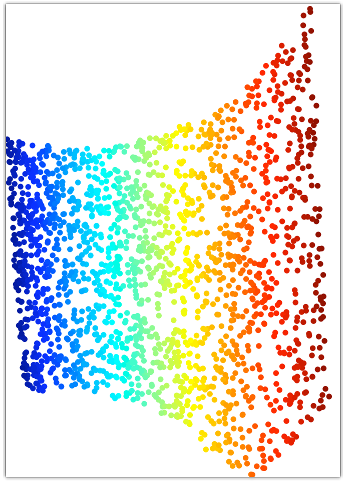
  Diagonalize the matrix



- **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

$$\phi : \boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$$

Perform PCA/MDS in the new space

$$\phi(\boldsymbol{X})^{\mathrm{T}} \phi(\boldsymbol{X}) \boldsymbol{v} = \lambda \boldsymbol{v}$$

(MDS: diagonlizing Gram matrix)

The kernel trick

The inner product

$$\phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$$

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

For certain mapping function, we can find a kernel function

$$\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$$

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 $\boldsymbol{K} = [K_{ij}] = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$

  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 would want to use kernels?

- **Handle complex data types.**

  Kernels for numerican data (eg., CPU load)

  $$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/\sigma\right)$$

  "String" kernels for text data (eg. 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**

$$\max \quad \sum_i \|\boldsymbol{y}_i\|^2$$ ← unfolding

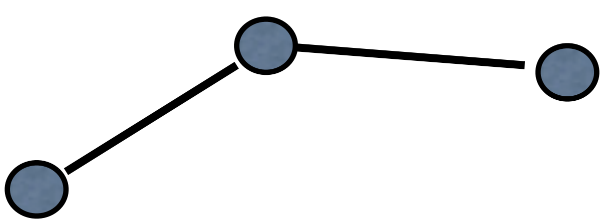
$$\sum_i \boldsymbol{y}_i = 0$$ ← centering

$$\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$$ ← only if i and j are nearest neighbor!

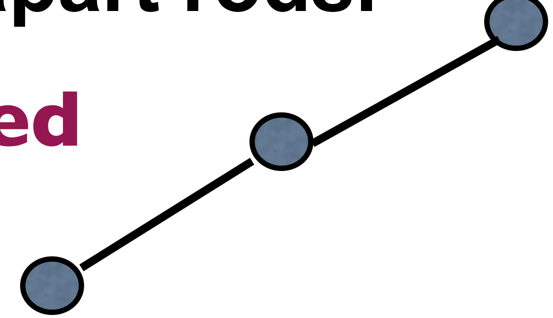
- **Intuition**

Nearby points are connected with rigid rods

Unfold inputs without breaking apart rods.



Rotation allowed →



Convex optimization

- **Change of variables**

$$K_{ij} = \boldsymbol{y}_i^{\mathrm{T}} \boldsymbol{y}_j$$

- **Semidefinite programming (SDP)**

$$\max \quad \sum_{ii} K_{ii}$$

unfolding objective

$$\sum_{ij} K_{ij} = 0$$

See this trick before?

$$K_{ii} + K_{jj} - 2K_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$$

$$\boldsymbol{K} \succeq 0$$

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

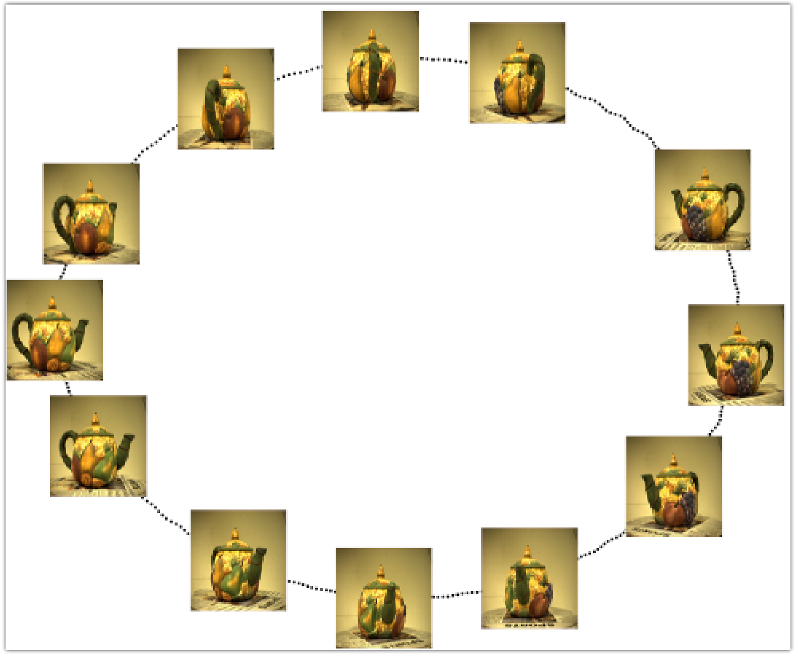
  Yield embedding and dimensionality

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

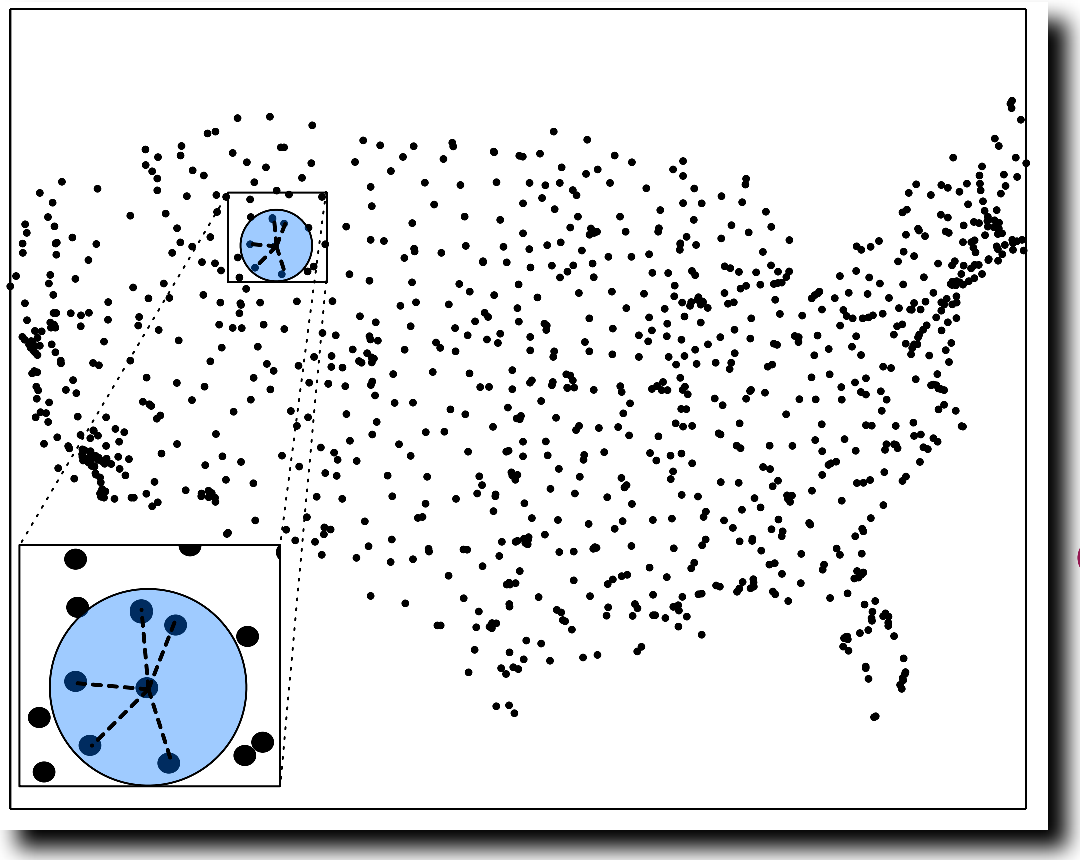
- **Differences**

  Semidefinite vs. dynamic programing to find Gram matrix

  Finite vs. asymptotic guarantee

  MVU works for manifolds with “holes”

Application: sensor localization



cities

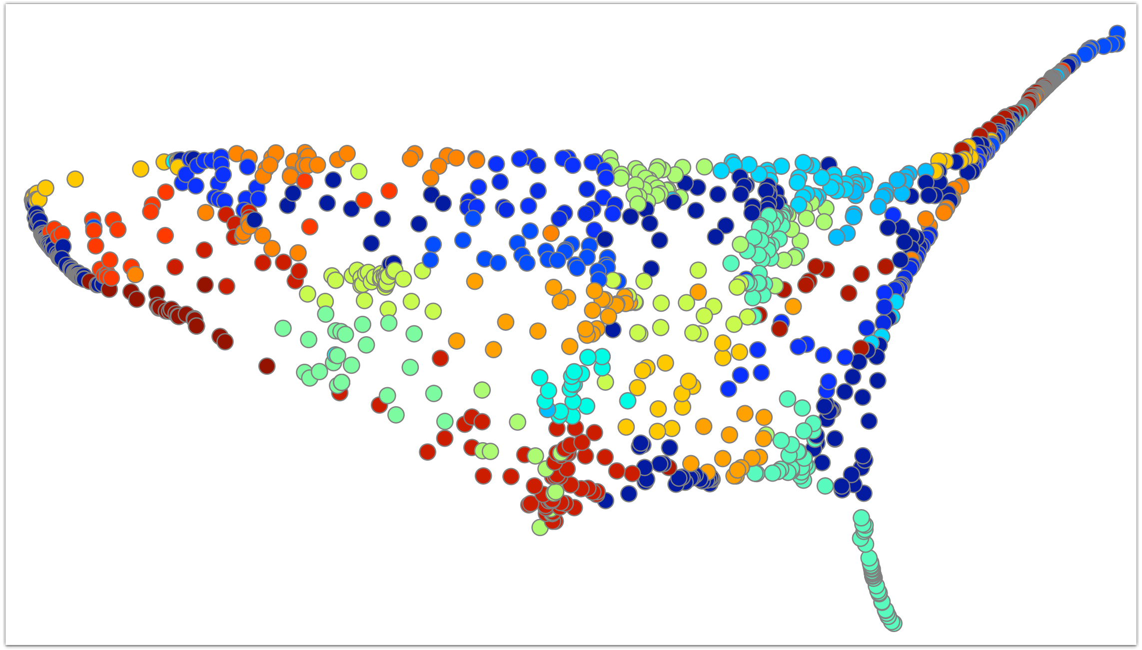
cities
$$\begin{bmatrix} 0 & d_{12} & ? & d_{14} \\ d_{21} & 0 & d_{23} & ? \\ ? & d_{32} & 0 & d_{34} \\ d_{41} & ? & d_{43} & 0 \end{bmatrix}$$

sensors distributed in US cities.

Infer coordinates from limited measurement of distances

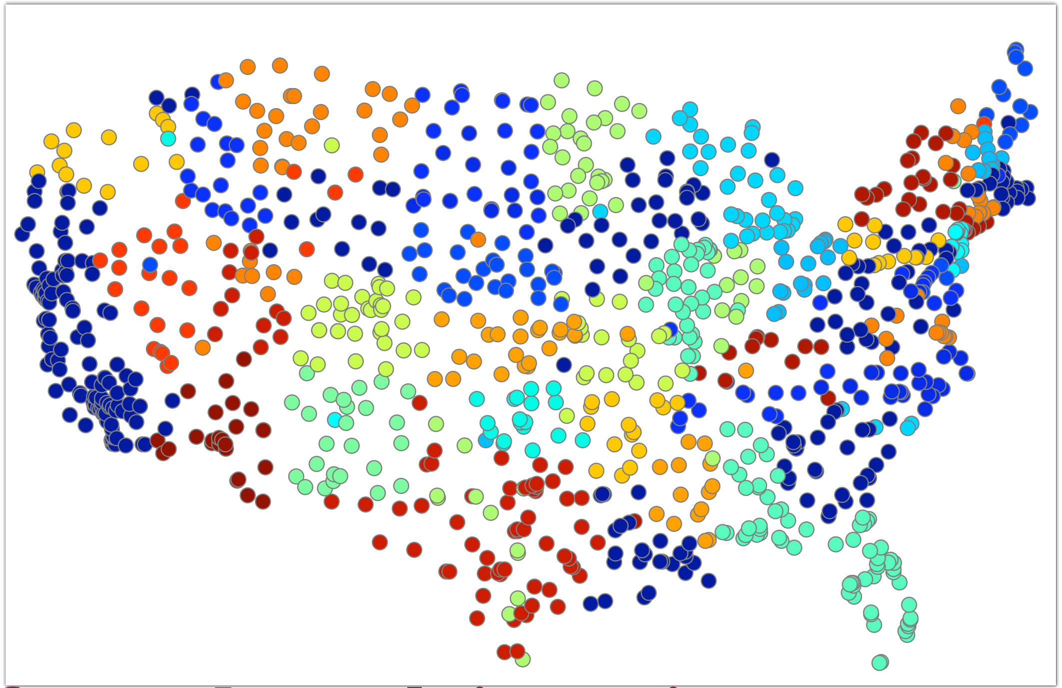
(Weinberger, Sha & Saul, NIPS 2006)

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 Lapalcian 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