Review

- Nonlinear dimensionality reduction (NLDR)
  Given high dimensional data that lies on a low-dimensional manifold, how to compute a faithful embedding?

- Notation: inputs $\vec{x}_i \in \mathbb{R}^D$
  outputs $\vec{y}_i \in \mathbb{R}^d$ with $d \ll D$

- Isomap algorithm
  **Step #1**: from inputs $\vec{x}_i \in \mathbb{R}^D$, construct neighborhood graph by linking $k$NN $\sim O(DN^2)$
  **Step #2**: compute shortest paths thru graph using dynamic programming (DP) $\sim O(N^3)$
  **Step #3**: from graph distances, compute outputs $\vec{y}_i \in \mathbb{R}^d$ using MDS $\sim O(dN^2)$

- Scaling to large data sets
  Problem: too expensive to compute all shortest paths and eigenvectors of full gram matrix
  Solution: only compute shortest paths for $n \times (N-n)$ slab of distance matrix $D_{ij}$ only compute eigenvectors for $nxn$ subblock of gram matrix $G_{ij}$
Nystrom approximation

Approximate \( G = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix} \) by \( \tilde{G} = \begin{pmatrix} A & B \\ B^T & B^T A^T B \end{pmatrix} \)

If \( \text{size}(A) \geq \text{rank}(G) \) and if \( A \) is full rank, then approximation is exact.

Why? Because rows are linearly dependent.

In practice, \( G \) is full rank, but dominated by few large eigenvalues.

Why? Because data is intrinsically low-dimensional.

Approximation not exact but highly accurate.

Timeline

2000: Isomap
2002: Laplacian eigenmaps
2004: Maximum variance unfolding (MVU)

Locally linear embedding (LLE)

Detour: semidefinite programming

Def: a semidefinite program (SDP) is a linear program with an extra constraint that a matrix whose elements are linear in the unknowns is positive semidefinite (PSD) with no negative eigenvalues.

Ex: \( X \) is unknown matrix

Maximize \( \text{tr}(AX) \) such that:

(i) \( \text{tr}(BiX) \geq C_i \) for \( i = 1, \ldots, \# \text{ constraints} \)

(ii) \( X \succeq 0 \) PSD constraint

Convex optimization

If \( X_1 \) and \( X_2 \) are both PSD, then so is \( \lambda X_1 + (1-\lambda)X_2 \) is also PSD for \( \lambda \in [0,1] \).

Efficient (poly-time) algorithms exist to solve SDPs (i.e. compute global optima).
Maximum variance unfolding

- **Outline**
  - Step #1 - compute kNN graph
  - Step #2 - "unfold" graph by solving SDP
  - Step #3 - apply MDS to "unfolded" graph

- **Comparison to Isomap**
  - same in steps #1 and #3
  - SDP vs. DP in step #2

**Intuition:**
To straighten a string, pull on its ends.
To flatten a sheet, pull on its four corners.
How does this idea extend to higher dimensions?

- **Quadratic programming (QP)**
  Maximize $\sum_i \|\vec{y}_i\|^2$ subject to:
  1. $\|\vec{y}_i - \vec{y}_j\|^2 = \|\vec{x}_i - \vec{x}_j\|^2$ if $\vec{x}_i$ and $\vec{x}_j$ are KNN
  2. $\sum_i \vec{y}_i = \vec{0}$

I.e. maximize variance of output subject to
(i) local distance and (ii) centering constraints.
Note: variance is bounded if kNN graph is connected.

- **Intuition**
  Connect KNN inputs by rigid rods.
Pull inputs apart without breaking rods.
Output find configuration.

- **Alternative formulations of distance constraints**
  $\|\vec{y}_i - \vec{y}_j\|^2 \leq \|\vec{x}_i - \vec{x}_j\|^2$ replace rods by strings
  $\|\vec{y}_i - \vec{y}_j\|^2 \geq \|\vec{x}_i - \vec{x}_j\|^2$ replace rods by springs

- **QP is hard to solve**
  Why? Maximizing (not minimizing) variance.
  Also: $\|\vec{y}_i - \vec{y}_j\|^2 = \|\vec{x}_i - \vec{x}_j\|^2$ not convex.
- Change of variables
  - Gram matrix $K_{ij} = \bar{y}_i \cdot \bar{y}_j$ determines outputs up to global rotation.
  - Replacing $\bar{y}_i \cdot \bar{y}_j$ by $K_{ij}$:
    \[
    \text{Variance} \quad \sum_i \| \bar{y}_i \|^2 = \sum_i K_{ii} = \text{tr}(K)
    \]
    \[
    \text{Distances} \quad \| \bar{y}_i - \bar{y}_j \|^2 = \| \bar{y}_i \|^2 + \| \bar{y}_j \|^2 - 2 \bar{y}_i \cdot \bar{y}_j = K_{ii} + K_{jj} - 2K_{ij}
    \]
    \[
    \text{Centracing} \quad \sum_i \bar{y}_i = 0 \implies \| (\sum_i \bar{y}_i) \|^2 = 0 \implies \sum_i \bar{y}_i \cdot \bar{y}_j = \sum_i K_{ij} = 0
    \]

- Relax QP to SDP
  - Maximize $\text{tr}(K)$ subject to:
    \[
    \begin{align*}
    (i) & \quad K_{ii} + K_{jj} - 2K_{ij} = \| x_i - x_j \|^2 \quad \text{if} \quad x_i \text{ and } x_j \text{ are kNN} \\
    (ii) & \quad \sum_j K_{ij} = 0 \\
    (iii) & \quad K \succeq 0 \quad (\text{PSD})
    \end{align*}
    \]
  - Constraint (iii) relaxing the implicit rank constraint $\bar{y}_i \in \mathbb{R}^d$ in QP.

- MVU versus PCA
  - PCA maximizes variance of linear projection.
  - MVU maximizes variance of nonlinear (but locally distance-preserving) projections.

- MVU versus Isomap
  - Motivated by isometry
  - Based on constructing Gram matrix
  - Eigenvalues resemble dimensionality

- Differences
  - SDP versus DP
  - Finite vs. asymptotic guarantees
  - Handling of manifolds with "holes"
Scaling

SDP scales at least as $O(n^3 + c^3)$ where

$n =$ size of matrix
$c =$ # constraints

Naive scaling for MVU is $O(k^3N^3)$ for $N$ examples and kNN.

How to improve?