CSE 291B 2/26

Locally linear embedding (LLE)

- Algorithm

  Step 1. Compute kNN

  Step 2. Compute local reconstruction weights W_{ij}.

  $$\min \|\vec{x}_i - \sum_j W_{ij}\vec{x}_j\|^2 \text{ such that } \begin{cases} W_{ij}=0 \text{ if } \vec{x}_j \text{ is not kNN of } \vec{x}_i \\ \sum_j W_{ij} = 1 \end{cases}$$

  Step 3. Compute outputs $\vec{y}_i \in \mathbb{R}^d$ that have same locally linear relationships.

  $$\min_Y \|\vec{y}_i - \sum_j W_{ij}\vec{y}_j\|^2 \text{ such that } \begin{cases} \sum_i \vec{y}_i = \vec{0} \\ \frac{1}{N}\sum_i \vec{y}_i\vec{y}_i^T = I_d \end{cases}$$

  Solution: bottom d eigenvectors of $(I-W)^T(I-W)$ with smallest non-zero eigenvalues

  $$Y = \left(\left(\vec{y}_1\right)\left(\vec{y}_2\right)\cdots\left(\vec{y}_N\right)\right) = \left(\begin{array}{l} \\ \\ \\ \\ \end{array}\right)$$ eigenvector 1 … eigenvector d

- More intuition for step 3.

  Thought experiment: after step 2, clamp subset of inputs ("landmarks") and reconstruct rest from weights W.

  $\min \|\vec{x}_i - \sum_j W_{ij}\vec{x}_j\|^2$ with respect to "non-clamped" inputs

  Cartoon picture

original $\vec{x}_i$ | N/2 clamped ∘, N/2 reconstructed × | N/4 clamped ∘, 3N/4 reconstructed ×

Reconstructions are highly accurate for large # landmarks. Increasing linearization occurs with decreasing # landmarks. Step #3 is limit with zero landmarks.

Recap of spectral methods

Algorithm	What is being preserved	Type of eigenvalue problem
MDS	euclidean distances	top eigenvectors of dense matrix
Isomap	geodesic distances	
MVS	local distances	
Laplacian eigenmaps	proximity	bottom eigenvectors of sparse matrix
LLE	local linear relationships	
Hessian LLE	⋮	
Local tangent space alignment		
Modified LLE		

Hybrid approach

- How to combine advantages of both families?
  - eigenspectrum that reveals d (dense)
  - scales well with N (not MVU)
- Essentials of MVU

  Let $K_{ij} = \vec{y}_i \cdot \vec{y}_j$

  Maximize $\mathrm{tr}(K)$ subject to:

  (i) $K_{ii} + K_{jj} - 2K_{ij} = \|\vec{x}_i - \vec{x}_j\|^2$ for kNN $\vec{x}_i, \vec{x}_j$

  (ii) $\sum_{ij} K_{ij} = 0$

  (iii) $K \succeq 0$

  But SDP over $N \times N$ matrix K scales poorly with N.
- Essentials of graph Laplacians

  $$W_{ij} = \begin{cases} \exp\left\{\frac{-1}{2\sigma^2}\|\vec{x}_i - \vec{x}_j\|^2\right\} & \text{if } \vec{x}_i, \vec{x}_j \text{ are kNN} \\ 0 & \text{otherwise} \end{cases}$$

  $$D_{ii} = \sum_j W_{ij}$$

  $$\mathcal{L} = I - D^{-1/2} W D^{-1/2}$$

  Eigenvectors of $\mathcal{L}$ are ordered basis for smooth functions on graph.

- "Laplacian" MVU

Let $\Phi \in \mathbb{R}^{m \times N}$ contain m (non-trivial) bottom eigenvectors of $\mathcal{L}$ with $m \ll N$ (say $m \sim 20$).

Note that MVU solution $Y \in \mathbb{R}^{m \times N}$ should be smooth with respect to graph.

To a good approximation $Y \approx L\Phi$ where $L \in \mathbb{R}^{m \times m}$

(Y: $m \times N$, L: $m \times m$, Φ: $m \times N$)

This implies $\underbrace{K}_{N \times N} = Y^T Y \approx \Phi^T L^T L \Phi$.

Rewrite MVU as optimization over matrix $P = L^T L \in \mathbb{R}^{m \times m}$ where $P \succeq 0$

This yields much smaller SDP

Kernel PCA

How to map $\vec{x}_i \in \mathbb{R}^D$ to $\vec{y}_i \in \mathbb{R}^d$?

- Idea: map inputs $\vec{x} \in \mathbb{R}^D$ to nonlinear feature vector $\Phi(\vec{x})$
  Then extract principal components in nonlinear feature space.
- Kernel "trick":
  Mapping $\vec{x} \longrightarrow \Phi(\vec{x})$ is implicitly specified by kernel function
  $$K(\vec{x}, \vec{x}') = \Phi(\vec{x}) \cdot \Phi(\vec{x}')$$
- Kernel matrix
  $K_{ij} = K(\vec{x}_i, \vec{x}_j)$ gram matrix in feature space
- Common kernel functions
  - polynomial $K(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^p$
    Maps $\vec{x} \to \Phi(\vec{x})$ of dimensionality $O(D^p)$

    Ex: $p=2$
    $$K(\vec{x}, \vec{x}') = (1 + \sum_\alpha x_\alpha x_\alpha')^2 = 1 + 2\sum_\alpha x_\alpha x_\alpha' + \sum_{\alpha\beta} x_\alpha x_\beta x_\alpha' x_\beta'$$
    suppose $\Phi(\vec{x}) = (1, \underbrace{\sqrt{2}x_\alpha}_{\substack{D \text{ elements} \\ \text{for } \alpha = 1..D}}, \underbrace{x_\alpha x_\beta}_{\substack{D^2 \text{ elements} \\ \alpha, \beta = 1..D}}) \in \mathbb{R}^{D^2+D+1}$
    Then: $\Phi(\vec{x}) \cdot \Phi(\vec{x}') = K(\vec{x}, \vec{x}')$

- gaussian kernel

  $K(\vec{x},\vec{x}') = e^{-\frac{1}{2\sigma^2}\|\vec{x}-\vec{x}\|^2}$

  Note: $k(\vec{x},\vec{x}) = \|\Phi(\vec{x})\|^2 = 1$ for all $\vec{x}$

  $\vec{x} \rightarrow \Phi(\vec{x})$ implicitly mapped to surface of infinite-dimensional sphere

- Centering transformation

  $\tilde{\Phi}(\vec{x}_i) = \Phi(\vec{x}_i) - \frac{1}{N}\sum_j \Phi(\vec{x}_j)$ subtracts out mean

  If $K_{ij} = k(\vec{x}_i,\vec{x}_j) = \Phi(\vec{x}_i)\,\Phi(\vec{x}_j)$, then

  $\tilde{K}_{ij} = \tilde{\Phi}(\vec{x}_i)\cdot\tilde{\Phi}(\vec{x}_j)$ is given by $\tilde{K} = (I-ee^T)K(I-ee^T)$

  with $e = \frac{1}{\sqrt{N}}(1,1,\ldots,1)$

- Exploit duality of MDS to PCA

  Difficult to compute covariance matrix $C = \frac{1}{N}\sum_i \tilde{\Phi}(\vec{x}_i)\tilde{\Phi}(\vec{x}_i)^T$

  Easy to compute centered kernel matrix $\tilde{K}$.

  But $\tilde{K}$ and C have same eigenvalues (up to multiplicative factor of N)

- Algorithm

  Compute top d eigenvalues $\{\lambda_\alpha\}$ and eigenvectors $\{\vec{v}_\alpha\}$ of $\tilde{K}$.

  Map $\vec{x}_i \in \mathbb{R}^D$ to $\vec{y}_i \in \mathbb{R}^d$ with $y_{\alpha i} = \sqrt{\lambda_\alpha}\, v_{\alpha i}$ $(\alpha = 1..d)$ just as in MDS.

  Estimate d from # dominant eigenvalues of $\tilde{K}$.

- For manifold learning?

  Pre-defined kernels (e.g. gaussian, polynomial) don't generally work for NLDR.

  How to learn kernel matrices (or functions) from high dimensional inputs?

  Isomap, MVU, LLE, etc. can be viewed as forms of kernel PCA with learned kernels.