Advanced Machine Learning & Perception

Instructor: Tony Jebara
Topic 2

- Nonlinear Manifold Learning
- Multidimensional Scaling (MDS)
- Locally Linear Embedding (LLE)
- Beyond Principal Components Analysis (PCA)
- Kernel PCA (KPCA)
- Semidefinite Embedding (SDE)
- Minimum Volume Embedding (MVE)
Principal Components Analysis

• Encode data on linear (flat) manifold as steps along its axes

$$\vec{x}_j \approx \vec{y}_j = \mu + \sum_{i=1}^{C} c_{ij} \vec{v}_i$$

• Best choice of $\mu$, $c$ and $v$ is least squares
  or equivalently maximum Gaussian likelihood

$$\text{error} = \sum_{j=1}^{N} \left\| \vec{x}_j - \vec{y}_j \right\|^2 = \sum_{j=1}^{N} \left\| \vec{x}_j - \mu - \sum_{i=1}^{C} c_{ij} \vec{v}_i \right\|^2$$

• Take derivatives of error over $\mu$, $c$ and $v$ and set to zero

$$\mu = \frac{1}{N} \sum_{j=1}^{N} \vec{x}_j, \quad v = eig\left\{\frac{1}{N} \sum_{j=1}^{N} (\vec{x}_j - \mu)(\vec{x}_j - \mu)^T\right\}, \quad c_{ij} = (\vec{x}_i - \mu)^T \vec{v}_j$$
Manifold Learning & Embedding

• Data is often not Gaussian and not in a linear subspace
• Consider image of face being translated from left-to-right

\[ \vec{x}_t = T^t \vec{x}_0 \]

• How to capture the true coordinates of the data on the manifold or embedding space and represent it compactly?
• Unlike PCA, Embedding does not try to reconstruct the data
• Just finds better more compact coordinates on the manifold
• Example, instead of pixel intensity image \((x, y)\) find a measure \((t)\) of how far the face has translated.
Multidimensional Scaling

• Idea: find low dimensional embedding that mimics only the **distances** between points X in original space
• Construct another set of low dimensional (say 2D) points with coordinates Y that maintain the pairwise distances
• A **Dissimilarity** $d(x_i, x_j)$ is a function of two inputs such that
  
  $d(\vec{x}_i, \vec{x}_j) \geq 0$
  
  $d(\vec{x}_i, \vec{x}_i) = 0$
  
  $d(\vec{x}_i, \vec{x}_j) = d(\vec{x}_j, \vec{x}_i)$

• A **Distance Metric** is stricter, satisfies triangle inequality:
  
  $d(\vec{x}_i, \vec{x}_k) \leq d(\vec{x}_i, \vec{x}_j) + d(\vec{x}_j, \vec{x}_k)$

• Standard example: Euclidean l2 metric $d(\vec{x}_i, \vec{x}_j) = \frac{1}{2} \| \vec{x}_i - \vec{x}_j \|^2$
• Assume for N objects, we compute a dissimilarity $\Delta$ matrix which tells us how far they are $\Delta_{ij} = d(\vec{x}_i, \vec{x}_j)$
Multidimensional Scaling

- Given dissimilarity $\Delta$ between original $X$ points under original $d()$ metric, find $Y$ points with dissimilarity $D$ under another $d'(\cdot)$ metric such that $D$ is similar to $\Delta$

$$ \Delta_{ij} = d(\vec{x}_i, \vec{x}_j) \quad D_{ij} = d'(\vec{y}_i, \vec{y}_j) $$

- Want to find $Y$'s that minimize some difference from $D$ to $\Delta$
- Eg. Least Squares Stress =
  $$ Stress(\vec{y}_1, \ldots, \vec{y}_N) = \sum_{ij} \left( D_{ij} - \Delta_{ij} \right)^2 $$
- Eg. Invariant Stress =
  $$ InvStress = \sqrt{\frac{Stress(Y)}{\sum_{i<j} D_{ij}^2}} $$
  Some are global
  Some are local
- Eg. Sammon Mapping =
  $$ \sum_{ij} \frac{1}{\Delta_{ij}} \left( D_{ij} - \Delta_{ij} \right)^2 $$
- Eg. Strain =
  $$ \text{trace} \left( J \left( \Delta^2 - D^2 \right) J \left( \Delta^2 - D^2 \right) \right) $$
  where $J = I - \frac{1}{N} \vec{1} \vec{1}^T$
MDS Example 3D to 2D

• Have distances from cities to cities, these are on the surface of a sphere (Earth) in 3D space
• Reconstructed 2D points on plane capture essential properties (poles?)

<table>
<thead>
<tr>
<th></th>
<th>London</th>
<th>Stockholm</th>
<th>Lisbon</th>
<th>Madrid</th>
<th>Paris</th>
<th>Amsterdam</th>
<th>Berlin</th>
<th>Prague</th>
<th>Rome</th>
<th>Dublin</th>
</tr>
</thead>
<tbody>
<tr>
<td>London</td>
<td>0</td>
<td>569</td>
<td>667</td>
<td>550</td>
<td>141</td>
<td>140</td>
<td>357</td>
<td>395</td>
<td>570</td>
<td>190</td>
</tr>
<tr>
<td>Stockholm</td>
<td>569</td>
<td>0</td>
<td>1212</td>
<td>1043</td>
<td>617</td>
<td>446</td>
<td>325</td>
<td>423</td>
<td>587</td>
<td>648</td>
</tr>
<tr>
<td>Lisbon</td>
<td>667</td>
<td>1212</td>
<td>0</td>
<td>201</td>
<td>596</td>
<td>768</td>
<td>923</td>
<td>882</td>
<td>714</td>
<td>914</td>
</tr>
<tr>
<td>Madrid</td>
<td>530</td>
<td>1043</td>
<td>201</td>
<td>0</td>
<td>431</td>
<td>608</td>
<td>940</td>
<td>690</td>
<td>516</td>
<td>622</td>
</tr>
<tr>
<td>Paris</td>
<td>141</td>
<td>617</td>
<td>596</td>
<td>431</td>
<td>0</td>
<td>177</td>
<td>340</td>
<td>337</td>
<td>436</td>
<td>320</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>140</td>
<td>446</td>
<td>768</td>
<td>608</td>
<td>177</td>
<td>0</td>
<td>218</td>
<td>272</td>
<td>519</td>
<td>302</td>
</tr>
<tr>
<td>Berlin</td>
<td>357</td>
<td>325</td>
<td>923</td>
<td>740</td>
<td>340</td>
<td>218</td>
<td>0</td>
<td>114</td>
<td>422</td>
<td>514</td>
</tr>
<tr>
<td>Prague</td>
<td>396</td>
<td>423</td>
<td>882</td>
<td>690</td>
<td>357</td>
<td>272</td>
<td>114</td>
<td>0</td>
<td>364</td>
<td>573</td>
</tr>
<tr>
<td>Rome</td>
<td>569</td>
<td>787</td>
<td>714</td>
<td>516</td>
<td>436</td>
<td>519</td>
<td>472</td>
<td>364</td>
<td>0</td>
<td>755</td>
</tr>
<tr>
<td>Dublin</td>
<td>190</td>
<td>648</td>
<td>714</td>
<td>622</td>
<td>320</td>
<td>302</td>
<td>514</td>
<td>573</td>
<td>755</td>
<td>0</td>
</tr>
</tbody>
</table>
MDS Example Multi-D to 2D

- More elaborate example
- Have correlation matrix between crimes. These are arbitrary dimensionality.
- Hack: convert correlation to dissimilarity and show reconstructed Y
Locally Linear Embedding

- Instead of trying to preserve ALL pairwise distances, only preserve SOME distances across nearby points:
  - Lets us unwrap manifold!
  - Also, distances are only locally valid
  - Euclidean distance is only similar to geodesic at small scales

- How do we pick which distances?
- Find the k nearest neighbors of each point and only preserve those
LLE with K-Nearest Neighbors

- Start with unconnected points

- Compute pairs of distances
  \[ A_{ij} = d(x_i, x_j) \]

- Connect each point to its k closest points
  \[ B_{ij} = A_{ij} \leq \text{sort}(A_{i*})_k \]

- Then symmetrize the connectivity matrix
  \[ B_{ij} = \max(B_{ji}, B_{ij}) \]
LLE

• Instead of distance, look at neighborhood of each point. Preserve reconstruction of point with neighbors in low dim.
• Find K nearest neighbors for each point
• Describe neighborhood as best weights on neighbors to reconstruct the point

$$\varepsilon(W) = \sum_i \left\| \hat{x}_i - \sum_j W_{ij} \vec{x}_j \right\|^2$$
subject to $$\sum_j W_{ij} = 1 \ \forall i$$

• Find best vectors that still have same weights

$$\Phi(Y) = \sum_i \left\| \vec{y}_i - \sum_j W_{ij} \vec{y}_j \right\|^2$$ subject to $$\sum_{i=1}^{N} \vec{y}_i = 0, \sum_{i=1}^{N} \vec{y}_i \vec{y}_i^T = I$$
LLE

Finding W’s (convex combination of weights on neighbors):

\[ \varepsilon(W) = \sum_i \varepsilon_i(W_i) \quad \text{where} \quad \varepsilon_i(W_i) = \| x_i - \sum_j W_{ij} x_j \|^2 \]

\[ \varepsilon_i(W_i) = \| \hat{x}_i - \sum_j W_{ij} \hat{x}_j \|^2 = \| \sum_j W_{ij} (\hat{x}_i - \hat{x}_j) \|^2 \]

\[ = \left( \sum_j W_{ij} (\hat{x}_i - \hat{x}_j) \right)^T \left( \sum_j W_{ij} (\hat{x}_i - \hat{x}_j) \right) \]

\[ = \sum_{jk} W_{ij} W_{ik} (\hat{x}_i - \hat{x}_j)^T (\hat{x}_i - \hat{x}_k) \]

\[ = \sum_{jk} W_{ij} W_{ik} C_{jk} \quad \text{and recall} \quad \sum_j W_{ij} = 1 \]

\[ W_i^* = \arg\min_w \frac{1}{2} w^T C w - \lambda \left( w^T \mathbf{1} \right) \]

1) Take Deriv & Set to 0

\[ Cw - \lambda \left( \mathbf{1} \right) = 0 \]

2) Solve Linear system

\[ C \left( \frac{w}{\lambda} \right) = \mathbf{1} \]

3) Find \( \lambda \)

\[ w^T \mathbf{1} = 1 \]

4) Find \( w \)

\[ \lambda \left( \frac{w}{\lambda} \right)^T \mathbf{1} = 1 \]
LLE

- Finding Y’s (new low-D points that agree with the W’s)

$$\Phi(Y) = \sum_i \left\| \vec{y}_i - \sum_j W_{ij} \vec{y}_j \right\|^2 \text{ subject to } \sum_{i=1}^N \vec{y}_i = 0, \sum_{i=1}^N \vec{y}_i \vec{y}_i^T = I$$

$$= \sum_i \left( \vec{y}_i - \sum_j W_{ij} \vec{y}_j \right) \left( \vec{y}_i - \sum_k W_{ik} \vec{y}_k \right)$$

$$= \sum_i \left( \vec{y}_i^T \vec{y}_i - \sum_k W_{ik} \vec{y}_i^T \vec{y}_k - \sum_j W_{ij} \vec{y}_j^T \vec{y}_i + \sum_{jk} W_{ij} W_{ik} \vec{y}_j^T \vec{y}_k \right)$$

$$= \sum_{jk} \left( \delta_{jk} - W_{jk} - W_{kj} + \sum_i W_{ij} W_{ik} \right) \vec{y}_j^T \vec{y}_k$$

$$= \sum_{jk} M_{jk} \vec{y}_j^T \vec{y}_k$$

$$= tr \left( MYY^T \right)$$

- Where Y is a matrix whose rows are the y vectors
- To minimize the above subject to constraints we set Y as the bottom d+1 eigenvectors of M
LLE Results

• Synthetic data on S-manifold
• Have noisy 3D samples X
• Get 2D LLE embedding Y

• Real data on face images
• Each x is an image that has been rasterized into a vector

• Dots are reconstructed two-dimensional Y points
LLE Results

- Top = PCA
- Bottom = LLE
Kernel Principal Components Analysis

• Recall, PCA approximates the data with eigenvectors, mean and coefficients: 
  \( \tilde{x}_i \approx \tilde{\mu} + \sum_{j=1}^C c_{ij} \tilde{v}_j \)

• Get eigenvectors that best approximating the covariance:
  \[ \Sigma = V \Lambda V^T \]

\[
\begin{bmatrix}
\Sigma_{11} & \Sigma_{12} & \Sigma_{13} \\
\Sigma_{12} & \Sigma_{22} & \Sigma_{23} \\
\Sigma_{13} & \Sigma_{23} & \Sigma_{33}
\end{bmatrix}
= 
\begin{bmatrix}
[\tilde{v}_1] & [\tilde{v}_2] & [\tilde{v}_3]
\end{bmatrix}
\begin{bmatrix}
\lambda_1 & 0 & 0 \\
0 & \lambda_2 & 0 \\
0 & 0 & \lambda_3
\end{bmatrix}
\begin{bmatrix}
[\tilde{v}_1] & [\tilde{v}_2] & [\tilde{v}_3]
\end{bmatrix}^T
\]

• Eigenvectors are orthonormal: \( \tilde{v}_i^T \tilde{v}_j = \delta_{ij} \)

• In coordinates of \( \nu \), Gaussian is diagonal, \( \text{cov} = \Lambda \)

• Higher eigenvalues are higher variance, use those first 
  \( \lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4 \geq \ldots \)

• To compute the coefficients: 
  \( c_{ij} = (\tilde{x}_i - \tilde{\mu})^T \tilde{v}_j \)

• How to extend PCA to make it nonlinear? Kernels!
Kernel PCA

- Idea: replace dot-products in PCA with kernel evaluations.
- Recall, could do PCA on DxD covariance matrix of data
  \[ C = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i \vec{x}_i^T \]
  \[ \lambda \vec{v} = C \vec{v} \]

  If data is zero-mean
  \[ \bar{C} = \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) \phi(x_i)^T \]

  or NxN Gram matrix of data: \[ K_{ij} = x_i^T x_j \]

  For nonlinearity, do PCA on feature expansions:

  \[ K_{ij} = k(x_i, x_j) = \phi(x_i)^T \phi(x_j) = (x_i^T x_j)^d \]

  • Instead of doing explicit feature expansion, use kernel
  I.e. d-th order polynomial

  • As usual, kernel must satisfy Mercer’s theorem

  • Assume, for simplicity, all feature data is zero-mean
  \[ \sum_{i=1}^{N} \phi(x_i) = 0 \]
Kernel PCA

- Efficiently find & use eigenvectors of $C$-bar: $\lambda \vec{v} = \vec{C} \vec{v}
- Can dot either side of above equation with feature vector:
  $$\lambda \phi(x_i)^T \vec{v} = \phi(x_i)^T \vec{C} \vec{v}$$
- Eigenvectors are in span of feature vectors: $\vec{v} = \sum_{i=1}^N \alpha_i \phi(x_i)$
- Combine equations:
  $$\lambda \phi(x_i)^T \vec{v} = \phi(x_i)^T \vec{C} \vec{v}$$
  $$\lambda \phi(x_i)^T \left\{ \sum_{j=1}^N \alpha_j \phi(x_j) \right\} = \phi(x_i)^T \vec{C} \left\{ \sum_{i=1}^N \alpha_j \phi(x_j) \right\}$$
  $$\lambda \phi(x_i)^T \left\{ \sum_{j=1}^N \alpha_j \phi(x_j) \right\} = \phi(x_i)^T \left\{ \frac{1}{N} \sum_{k=1}^N \phi(x_k) \phi(x_k)^T \right\} \left\{ \sum_{j=1}^N \alpha_j \phi(x_j) \right\}$$
  $$\lambda \sum_{j=1}^N \alpha_j K_{ij} = \frac{1}{N} \sum_{k=1}^N K_{ik} \sum_{j=1}^N \alpha_j K_{kj}$$
  $$N \lambda \vec{k} \vec{\alpha} = \vec{K}^2 \vec{\alpha}$$
  $$N \lambda \vec{\alpha} = \vec{K} \vec{\alpha}$$
Kernel PCA

• From before, we had: 
  \[ \lambda \phi(x_i)^T \bar{v} = \phi(x_i)^T \bar{C} \bar{v} \]
  this is an eig equation!
• Get eigenvectors \( \alpha \) and eigenvalues of \( K \)
• Eigenvalues are \( N \) times \( \lambda \)
• For each eigenvector \( \alpha^k \) there is an eigenvector \( v^k \)
• Want eigenvectors \( v \) to be normalized:
  \[ \left( \bar{v}^k \right)^T \bar{v}^k = 1 \]

\[
\left( \sum_{i=1}^{N} \alpha^k_i \phi(x_i) \right)^T \left( \sum_{j=1}^{N} \alpha^k_j \phi(x_j) \right) = 1
\]

\[
\left( \bar{\alpha}^k \right)^T K \bar{\alpha}^k = 1
\]

\[
\left( \bar{\alpha}^k \right)^T N \lambda^k \bar{\alpha}^k = 1
\]

\[
\bar{\alpha}^k \bar{\alpha}^k = \frac{1}{N \lambda^k}
\]

• Can now use alphas only for doing PCA projection & reconstruction!
Kernel PCA

- To compute k'th projection coefficient of a new point $\phi(x)$
  \[ c_k^k = \phi(x)^T \tilde{v}_k = \phi(x)^T \left\{ \sum_{i=1}^N \alpha_i^k \phi(x_i) \right\} = \sum_{i=1}^N \alpha_i^k k(x, x_i) \]

- Reconstruction*:
  \[ \tilde{\phi}(x) = \sum_{k=1}^K c_k^k \tilde{v}_k = \sum_{k=1}^K \sum_{i=1}^N \alpha_i^k k(x, x_i) \sum_{j=1}^N \alpha_j^k \phi(x_j) \]

  *Pre-image problem, linear combo in Hilbert goes outside

- Can now do nonlinear PCA and do PCA on non-vectors

- Nonlinear KPCA eigenvectors satisfy same properties as usual PCA but in Hilbert space. These evecs:
  1) Top q have max variance
  2) Top q reconstruction has with min mean square error
  3) Are uncorrelated/orthogonal
  4) Top have max mutual with inputs
Centering Kernel PCA

• So far, we had assumed the data was zero-mean: 
  \[ \sum_{i=1}^{N} \phi(x_i) = 0 \]
• We want this: 
  \[ \tilde{\phi}(x_j) = \phi(x_j) - \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) \]
• How to do without touching feature space? Use kernels...
  \[ \tilde{K}_{ij} = \tilde{\phi}(x_i)^T \tilde{\phi}(x_j) \]
  \[
  = \left( \phi(x_i) - \frac{1}{N} \sum_{k=1}^{N} \phi(x_k) \right)^T \left( \phi(x_j) - \frac{1}{N} \sum_{k=1}^{N} \phi(x_k) \right)
  
  = \phi(x_i)^T \phi(x_j) - \frac{1}{N} \sum_{k=1}^{N} \phi(x_k)^T \phi(x_j)
  
  - \frac{1}{N} \sum_{k=1}^{N} \phi(x_i)^T \phi(x_k) + \frac{1}{N} \frac{1}{N} \sum_{k=1}^{N} \sum_{l=1}^{N} \phi(x_k)^T \phi(x_l)
  
  = K_{ij} - \frac{1}{N} \sum_{k=1}^{N} K_{kj} - \frac{1}{N} \sum_{k=1}^{N} K_{ik} + \frac{1}{N^2} \sum_{k=1}^{N} \sum_{l=1}^{N} K_{kl}
  \]

• Can get alpha eigenvectors from K tilde by adjusting old K
Kernel PCA Results

• KPCA on 2d dataset

• Left-to-right Kernel poly order goes from 1 to 3

1=linear=PCA

• Top-to-bottom top evec to weaker evecs

Figure 2: Two-dimensional toy examples, with data generated in the following way: \( x \)-values have uniform distribution in \([-1, 1]\), \( y \)-values are generated from \( y_t = x_t^2 + \xi \), were \( \xi \) is normal noise with standard deviation 0.2. From left to right, the polynomial degree in the kernel (22) increases from 1 to 4; from top to bottom, the first 3 Eigenvectors are shown (in order of decreasing Eigenvalue size). The figures contain lines of constant principal component value (contour lines); in the linear case, these are orthogonal to the Eigenvectors. We did not draw the Eigenvectors, as in the general case, they live in a higher-dimensional space. Note that linear PCA only leads to 2 nonzero Eigenvalues, as the input dimensionality is 2. In contrast, nonlinear PCA uses the third component to pick up the variance caused by the noise, as can be seen in the case of degree 2.
Kernel PCA Results

- Use coefficients of the KPCA for training a linear SVM classifier to recognize chairs from their images.
- Use various polynomial kernel degrees where $1=\text{linear}$ as in regular PCA.

<table>
<thead>
<tr>
<th># of components</th>
<th>Test Error Rate for degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>23.0</td>
</tr>
<tr>
<td>128</td>
<td>17.6</td>
</tr>
<tr>
<td>256</td>
<td>16.8</td>
</tr>
<tr>
<td>512</td>
<td>n.a.</td>
</tr>
<tr>
<td>1024</td>
<td>n.a.</td>
</tr>
<tr>
<td>2048</td>
<td>n.a.</td>
</tr>
</tbody>
</table>

Table 1: Test error rates on the MPI chair database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with kernel (22), for degrees 1 through 7. In the case of degree 1, we are doing standard PCA, with the number of nonzero Eigenvalues being at most the dimensionality of the space, 256; thus, we can extract at most 256 principal components. The performance for the nonlinear cases (degree $>1$) is significantly better than for the linear case, illustrating the utility of the extracted nonlinear components for classification.
Kernel PCA Results

- Use coefficients of the KPCA for training a linear SVM classifier to recognize characters from their images.
- Use various polynomial kernel degrees where 1=linear as in regular PCA (worst case in experiments)
- Inferior performance to nonlinear SVMs (why??)

<table>
<thead>
<tr>
<th># of components</th>
<th>Test Error Rate for degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>9.6</td>
</tr>
<tr>
<td>64</td>
<td>8.8</td>
</tr>
<tr>
<td>128</td>
<td>8.6</td>
</tr>
<tr>
<td>256</td>
<td>8.7</td>
</tr>
<tr>
<td>512</td>
<td>n.a.</td>
</tr>
<tr>
<td>1024</td>
<td>n.a.</td>
</tr>
<tr>
<td>2048</td>
<td>n.a.</td>
</tr>
</tbody>
</table>

Table 2: Test error rates on the USPS handwritten digit database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with kernel (22), for degrees 1 through 7. In the case of degree 1, we are doing standard PCA, with the number of nonzero Eigenvalues being at most the dimensionality of the space, 256; thus, we can extract at most 256 principal components. Clearly, nonlinear principal components afford test error rates which are superior to the linear case (degree 1).
Semidefinite Embedding

• Also known as Maximum Variance Unfolding
• Similar to LLE and kernel PCA
• Like LLE, maintains only distance in the neighborhood
• Stretch all the data while maintaining the distances:

• Then apply PCA (or kPCA)
Semidefinite Embedding

• To visualize high-dimensional \( \{x_1, \ldots, x_N\} \) data:

• PCA and Kernel PCA (Sholkopf et al):
  - Get matrix \( A \) of affinities between pairs \( A_{ij} = k(x_i, x_j) \)
  - SVD \( A \) & view top projections

• Semidefinite Embedding (Weinberger, Saul):
  - Get k-nearest neighbors graph of data
  - Get matrix \( A \)
  - Use max trace SDP to stretch
    stretch graph \( A \) into PD graph \( K \)
  - SVD \( K \) & view top projections
Semidefinite Embedding

- SDE unfolds (pulls apart) knn connected graph $C$ but preserves pairwise distances when $C_{ij} = 1$

$$\max_K \sum_i \lambda_i \quad s.t. \quad K \in \kappa$$

$$\kappa = \forall K \in \mathbb{R}^{N \times N}$$

$$s.t. \quad K \succeq 0$$

$$s.t. \quad \sum_{ij} K_{ij} = 0$$

$$s.t. \quad K_{ii} + K_{jj} - K_{ij} - K_{ji} =$$

$$A_{ii} + A_{jj} - A_{ij} - A_{ji} \quad if \quad C_{ij} = 1$$

- SDE’s stretching of graph *improves* the visualization
SDE Optimization with YALMIP

Linear Programming
  < Quadratic Programming
    < Quadratically Constrained Quadratic Programming
      < Semidefinite Programming
        < Convex Programming
          < Polynomial Time Algorithms

LP  QP  QCQP  SDP  CP  P
SDE Optimization with YALMIP

- **LP**
  \[
  \min_x \bar{b}^T \bar{x} \quad s.t. \quad \bar{c}_i^T \bar{x} \geq \alpha_i \quad \forall i
  \]
  \(\text{fast } O(N^3)\)

- **QP**
  \[
  \min_x \frac{1}{2} \bar{x}^T H \bar{x} + \bar{b}^T \bar{x} \quad s.t. \quad \bar{c}_i^T \bar{x} \geq \alpha_i \quad \forall i
  \]

- **QCQP**
  \[
  \min_x \frac{1}{2} \bar{x}^T H \bar{x} + \bar{b}^T \bar{x} \quad s.t. \quad \bar{c}_i^T \bar{x} \geq \alpha_i \quad \forall i, \quad \bar{x}^T \bar{x} \leq \eta
  \]

- **SDP**
  \[
  \min_K tr(BK) \quad s.t. \quad tr(C_i^T K) \geq \alpha_i \quad \forall i, \quad K \succeq 0
  \]

... ALL above in the YALMIP package for Matab!

... Google it, download and install!

- **CP**
  \[
  \min_x f(\bar{x}) \quad s.t. \quad g(\bar{x}) \geq \alpha
  \]
  \(\text{slow } O(N^3)\)
SDE Results

• SDE unfolds (pulls apart) knn connected graph C

• SDE’s stretching of graph *improves* the visualization here
SDE Results

• SDE unfolds (pulls apart) knn connected graph C before doing PCA

• Gets more use or energy out of the top eigenvectors than PCA

• SDE’s stretching of graph improves visualization here
SDE Problems $\rightarrow$ MVE

• But SDE stretching could *worsen* visualization!

• Spokes Experiment:

  • Want to pull apart only in visualized dimensions
  • Flatten down remaining ones
Minimum Volume Embedding

- To reduce dimension, drive energy into top (e.g. 2) dims
- Maximize fidelity $F(K) = \frac{\lambda_1 + \lambda_2}{\sum_i \lambda_i}$ or % energy in top dims

Minimum Volume Embedding

$F(K) = 0.98$

$F(K) = 0.29$

- Equivalent to maximizing $\lambda_1 + \lambda_2 - \beta \sum_i \lambda_i$ for some $\beta$
- Assume $\beta = 1/2...$
Minimum Volume Embedding

• Stretch in d<D top dimensions and squash rest.

\[
\max_K \sum_{i=1}^{d} \lambda_i - \sum_{i=d+1}^{D} \lambda_i \quad \text{s.t.} \quad K \in \mathcal{K}
\]

• Simplest Linear-Spectral SDP...

\[
\tilde{\alpha} = \begin{bmatrix}
\alpha_1 & \cdots & \alpha_d & \alpha_{d+1} & \cdots & \alpha_D \\
+1 & \cdots & +1 & -1 & \cdots & -1
\end{bmatrix}
\]

• Effectively maximizes Eigengap between d’th and d+1’th \( \lambda \)
Minimum Volume Embedding

• Stretch in d < D top dimensions and **squash** rest.

\[
\max_K \sum_{i=1}^{d} \lambda_i - \sum_{i=d+1}^{D} \lambda_i \quad s.t. \quad K \in \kappa
\]

• Simplest Linear-Spectral SDP...

\[
\tilde{\alpha} = \begin{bmatrix}
\alpha_1 & \cdots & \alpha_d & \alpha_{d+1} & \cdots & \alpha_D \\
1 & \cdots & 1 & -1 & \cdots & -1
\end{bmatrix}
\]

• Effectively maximizes **Eigengap**
  between d’th and d+1’th \(\lambda\)

• Variational bound on cost \(\rightarrow\) Iterated Monotonic SDP
• Lock V and solve SDP K.
  Lock K and solve SVD for V.
MVE Optimization: Spectral SDP

- Spectral function: \( f(\lambda_1, \ldots, \lambda_D) \) eigenvalues of a matrix \( K \)
- SDP packages use restricted cost functions over hull kappa.

Trace SDPs

\[
\max_{K \in \kappa} \text{tr}(BK)
\]

Logdet SDPs

\[
\max_{K \in \kappa} \sum_i \log \lambda_i
\]

Consider richer SDPs (assume \( \lambda_i \) in decreasing order)

Linear-Spectral SDPs

\[
\max_{K \in \kappa} \sum_i \alpha_i \lambda_i
\]

Problem: last one doesn’t fit nicely into standard SDP code (like YALMIP or CSDP). Need an iterative algorithm...
MVE Optimization: Spectral SDP

• If alphas are ordered get a variational SDP problem (a Procrustes problem).

\[
\max_{K \in \mathbb{K}} g(K) = \max_{K \in \mathbb{K}} \sum_i \alpha_i \lambda_i \\
= \max_{K \in \mathbb{K}} \sum_i \alpha_i \lambda_i \text{tr}(v_i^T v_i) \\
= \max_{K \in \mathbb{K}} \sum_i \alpha_i \text{tr}(\lambda_i v_i v_i^T) \\
= \max_{K \in \mathbb{K}} \sum_i \alpha_i \text{tr}(K v_i v_i^T) \\
= \max_{K \in \mathbb{K}} \text{tr}(K \sum_i \alpha_i v_i v_i^T)
\]

\[
\begin{align*}
\max_{K \in \mathbb{K}} \min_U & \quad \text{tr} \left( K \sum_i \alpha_i u_i u_i^T \right) \quad \text{s.t.} \quad u_i^T u_j = \delta_{ij} \quad \text{if} \quad \alpha_i \leq \alpha_{i+1} \\
\max_{K \in \mathbb{K}} \max_U & \quad \text{tr} \left( K \sum_i \alpha_i u_i u_i^T \right) \quad \text{s.t.} \quad u_i^T u_j = \delta_{ij} \quad \text{if} \quad \alpha_i \geq \alpha_{i+1}
\end{align*}
\]

For max over \( K \) use SDP. For max over \( U \) use SVD. Iterate to obtain monotonic improvement.
MVE Optimization: Spectral SDP

• Theorem: if alpha decreasing the objective

\[ g(K) = \sum_i \alpha_i \lambda_i \quad \text{s.t.} \quad \alpha_i \geq \alpha_{i+1} \]

is convex.

• Proof: Recall from (Overton & Womersley ’91) and (Fan ’49) and (Bach & Jordan ’03)

"Sum of d top eigenvalues of p.d. matrix is convex"

\[ f_d(K) = \sum_{i=1}^d \lambda_i \Rightarrow \text{convex} \]

Our linear-spectral cost is a combination of these

\[ g(K) = \alpha_D f_D(K) + \sum_{i=D-1}^1 (\alpha_i - \alpha_{i+1}) f_i(K) \]

\[ = \alpha_D \text{tr}(K) + \sum_{i=D-1}^1 \left| \alpha_i - \alpha_{i+1} \right| f_i(K) \]

Trace (linear) + conic combo of convex fn’s = convex
### MVE Pseudocode

- Download at [www.metablake.com/mve](http://www.metablake.com/mve)

<table>
<thead>
<tr>
<th>Input</th>
<th>((\mathbf{x}<em>i)</em>{i=1}^N), kernel (\kappa), and parameters (d, k).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Form affinity matrix (A \in \mathbb{R}^{N \times N}) with pairwise entries (A_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)).</td>
</tr>
<tr>
<td>Step 2</td>
<td>Use (A) to find a binary connectivity matrix (C) via (k)-nearest neighbors.</td>
</tr>
<tr>
<td>Step 3</td>
<td>Initialize (\hat{K} = A).</td>
</tr>
<tr>
<td>Step 4</td>
<td>Solve for the eigenvectors (\mathbf{v}_1, \ldots, \mathbf{v}_N) and eigenvalues (\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N) of (K).</td>
</tr>
<tr>
<td>Step 5</td>
<td>Set (B = -\sum_{i=1}^{d} \mathbf{v}_i \mathbf{v}<em>i^T + \sum</em>{i=d+1}^{N} \mathbf{v}_i \mathbf{v}_i^T).</td>
</tr>
<tr>
<td>Step 6</td>
<td>Using SDP find (\hat{K} = \arg \min_{K \in \mathcal{K}} \text{tr}(KB)).</td>
</tr>
<tr>
<td>Step 7</td>
<td>If (|K - \hat{K}| \geq \epsilon) set (K = \hat{K}), go to Step 4.</td>
</tr>
<tr>
<td>Step 8</td>
<td>Perform kernel PCA on (K^*) to get (d)-dimensional output vectors (\mathbf{y}_1, \ldots, \mathbf{y}_N).</td>
</tr>
</tbody>
</table>
MVE Results

- Spokes experiment visualization and spectra
- Converges in ~5 iterations
MVE Results

• Swissroll Visualization (Connectivity via knn) (d is set to 2)

• Same convergence under random initialization or $K=A...$
kNN Embedding with MVE

- MVE does better even with kNN connectivity
- Face images with spectra
kNN Embedding with MVE

- MVE does better even with kNN
- Digit images visualization with spectra
Graph Embedding with MVE

- Collaboration network with spectra

PCA

SDE

MVE

5.9% in 2D

95.3% in 2D

99.2% in 2D
MVE Results

- Evaluate fidelity or % energy in top 2 dims

<table>
<thead>
<tr>
<th></th>
<th>kPCA</th>
<th>SDE</th>
<th>MVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star</td>
<td>95.0%</td>
<td>29.9%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Swiss Roll</td>
<td>45.8%</td>
<td>99.9%</td>
<td>99.9%</td>
</tr>
<tr>
<td>Twos</td>
<td>18.4%</td>
<td>88.4%</td>
<td>97.8%</td>
</tr>
<tr>
<td>Faces</td>
<td>31.4%</td>
<td>83.6%</td>
<td>99.2%</td>
</tr>
<tr>
<td>Network</td>
<td>5.9%</td>
<td>95.3%</td>
<td>99.2%</td>
</tr>
</tbody>
</table>
MVE for Spanning Trees

• Instead of kNN, use maximum weight spanning tree to connect points

(Kruskal’s algo: connect points with short edges first, skip edges that create loops, stop when tree)

• Tree connectivity can fold under SDE or MVE.
• Add constraints on all pairs to keep all distances from shrinking (call this SDE-FULL or MVE-FULL)

\[ K \in \text{original} \kappa \]

\[ \text{and } K_{ii} + K_{jj} - K_{ij} - K_{ji} \geq A_{ii} + A_{jj} - A_{ij} - A_{ji} \]
MVE for Spanning Trees

- Tree connectivity with degree=2.
- Top: taxonomy tree of 30 species of salamanders
- Bottom: taxonomy tree of 56 species of crustaceans