Advanced Machine Learning & Perception

Instructor: Tony Jebara

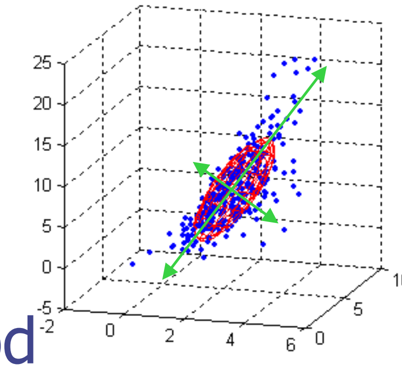
Topic 2

- 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

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



- Best choice of μ, c and v is least squares or equivalently maximum Gaussian likelihood

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

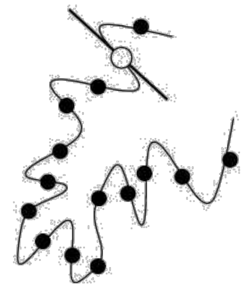
- Take derivatives of error over μ, c and v and set to zero

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

Manifold Learning & Embedding

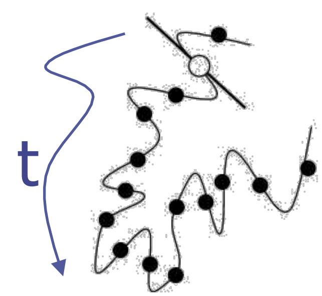
- Data is often not Gaussian and not in a linear subspace
- Consider image of face being translated from left-to-right

$$\vec{x}_t = T^t \vec{x}_0$$



...nonlinear!

- 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

$$d\left(\vec{x}_i,\vec{x}_j\right) \geq 0$$

$$d\left(\vec{x}_i,\vec{x}_i\right) = 0$$

$$d\left(\vec{x}_i,\vec{x}_j\right) = d\left(\vec{x}_j,\vec{x}_i\right)$$

- A Distance Metric is stricter, satisfies triangle inequality:

$$d\left(\vec{x}_i,\vec{x}_k\right) \leq d\left(\vec{x}_i,\vec{x}_j\right) + d\left(\vec{x}_j,\vec{x}_k\right)$$

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

Multidimensional Scaling

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

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

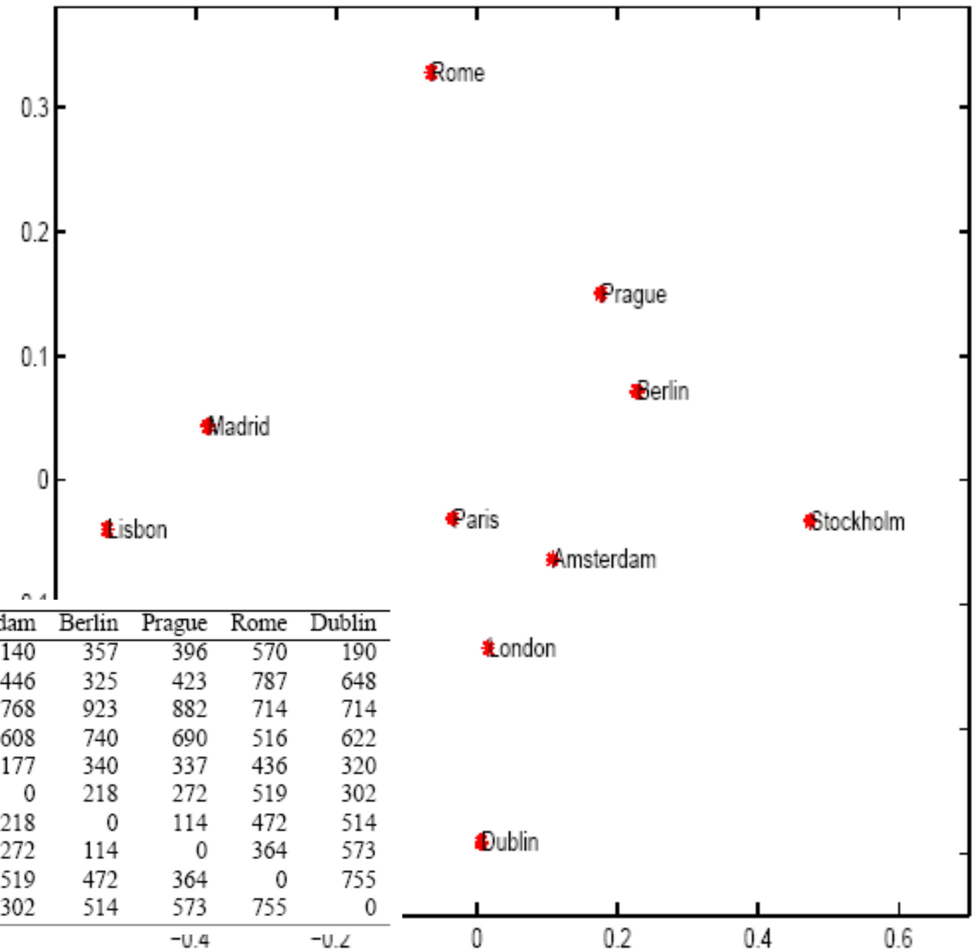
- Want to find Y's that minimize some difference from D to Δ
- Eg. Least Squares Stress = $Stress\left(\vec{y}_1, \ldots, \vec{y}_N\right) = \sum_{ij} \left(D_{ij} - \Delta_{ij}\right)^2$
- Eg. Invariant Stress = $InvStress = \sqrt{\frac{Stress\left(Y\right)}{\sum_{i<j} D_{ij}^2}}$
- Eg. Sammon Mapping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

Some are global
Some are local
Gradient descent

Tony Jebara, Columbia University

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?)

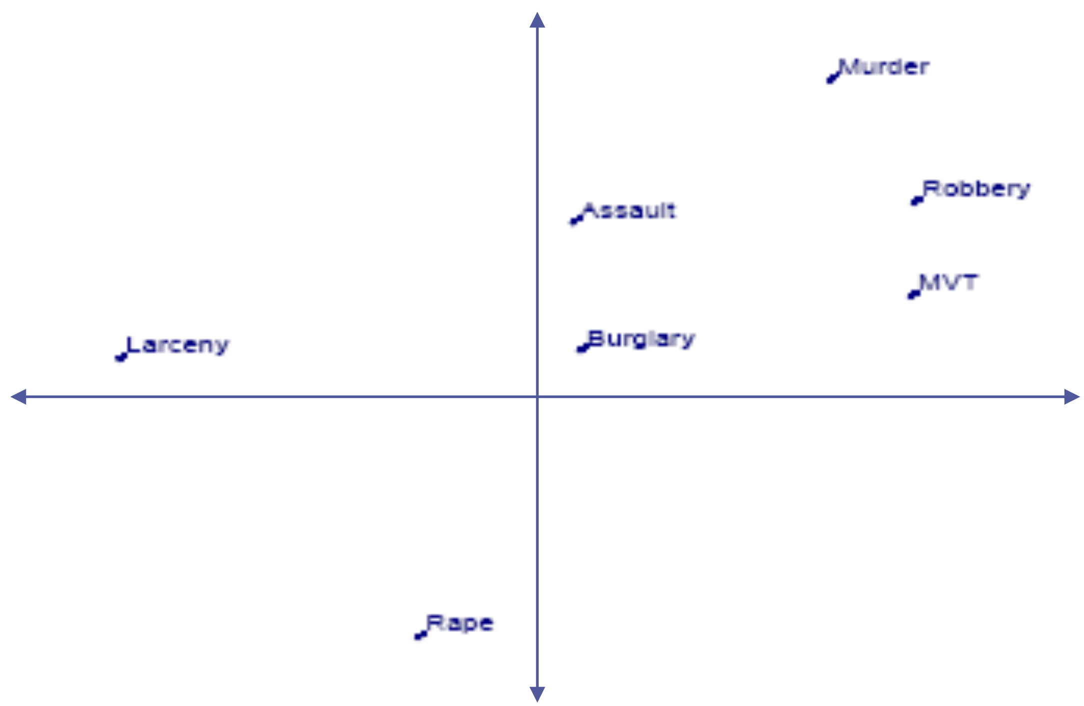


	London	Stockholm	Lisbon	Madrid	Paris	Amsterdam	Berlin	Prague	Rome	Dublin
London	0	569	667	530	141	140	357	396	570	190
Stockholm	569	0	1212	1043	617	446	325	423	787	648
Lisbon	667	1212	0	201	596	768	923	882	714	714
Madrid	530	1043	201	0	431	608	740	690	516	622
Paris	141	617	596	431	0	177	340	337	436	320
Amsterdam	140	446	768	608	177	0	218	272	519	302
Berlin	357	325	923	740	340	218	0	114	472	514
Prague	396	423	882	690	337	272	114	0	364	573
Rome	569	787	714	516	436	519	472	364	0	755
Dublin	190	648	714	622	320	302	514	573	755	0

MDS Example Multi-D to 2D

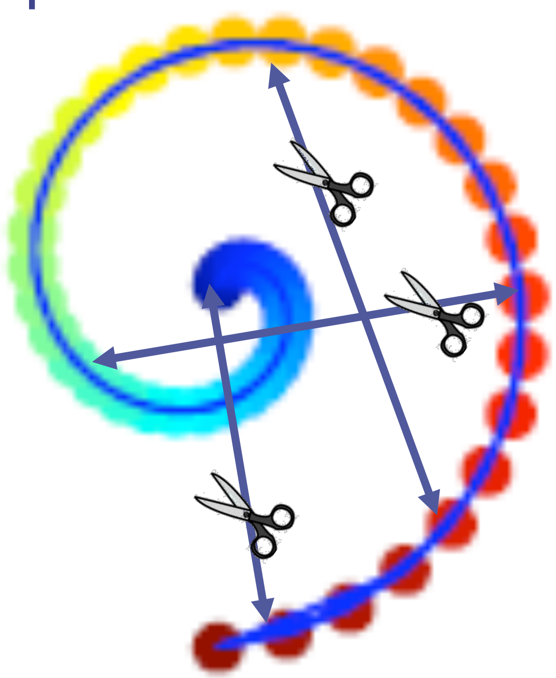
- More elaborate example
- Have correlation matrix between crimes. These are arbitrary dimensionality.
- Hack: convert correlation to dissimilarity and show reconstructed Y

	Murder	Rape	Robbery	Assault	Burglary	Larceny	MVT
Murder	1.000000	4.424527	1.430246	1.991164	1.949596	6.0901055	2.090254
Rape	4.424527	1.000000	4.184025	3.403713	1.930864	3.3641742	5.644764
Robbery	1.430246	4.184025	1.000000	1.513991	1.677549	6.5831954	1.417225
Assault	1.991164	3.403713	1.513991	1.000000	1.466635	1.9557311	1.738007
Burglary	1.949596	1.930864	1.677549	1.466635	1.000000	1.6972866	1.732629
Larceny	6.090106	3.364174	6.583195	1.955731	1.697287	1.0000000	4.614750
MVT	2.090254	5.644764	1.417225	1.738007	1.732629	4.6147505	1.000000



Locally Linear Embedding

- Instead of trying to preserve ALL pairwise distances only preserve SOME distances across nearby points:



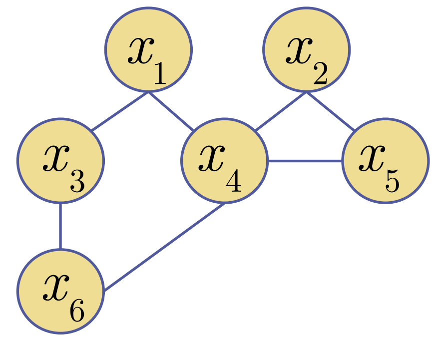
Lets us unwrap manifold!
Also, distances are
only locally valid

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
  $B_{ij} = A_{ij} <= \mathrm{sort}(A_{i*})_k$

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

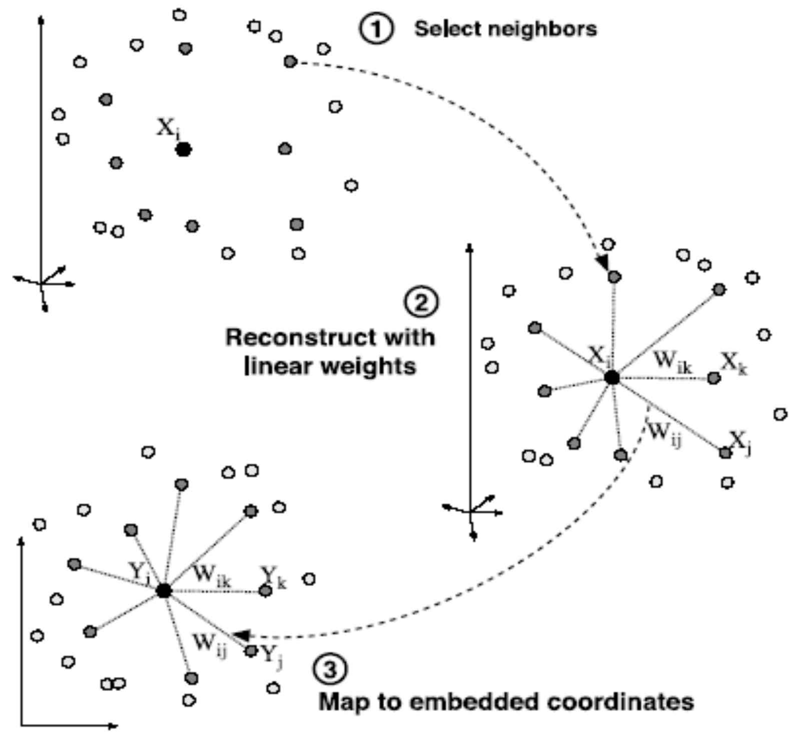
	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	1	1	0	0
x_2	0	0	0	1	1	0
x_3	1	0	0	0	0	1
x_4	1	1	0	0	0	0
x_5	0	1	0	1	0	0
x_6	0	0	1	1	0	0

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

$$\varepsilon(W) = \sum_i \left\| \vec{x}_i - \sum_j W_{ij} \vec{x}_j \right\|^2$$

$$subject\,to \sum_j W_{ij} = 1 \;\; \forall\, i$$



- 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):

$$\varepsilon(W) = \sum_i \varepsilon_i(W_{i\bullet}) \quad where\ \varepsilon_i(W_{i\bullet}) = \left\|\vec{x}_i - \sum_j W_{ij}\vec{x}_j\right\|^2$$

$$\varepsilon_i(W_{i\bullet}) = \left\|\vec{x}_i - \sum_j W_{ij}\vec{x}_j\right\|^2 = \left\|\sum_j W_{ij}\left(\vec{x}_i - \vec{x}_j\right)\right\|^2$$

$$= \left(\sum_j W_{ij}\left(\vec{x}_i - \vec{x}_j\right)\right)^T \left(\sum_j W_{ij}\left(\vec{x}_i - \vec{x}_j\right)\right)$$

$$= \sum_{jk} W_{ij}W_{ik}\left(\vec{x}_i - \vec{x}_j\right)^T\left(\vec{x}_i - \vec{x}_k\right)$$

$$= \sum_{jk} W_{ij}W_{ik}C_{jk} \quad and\ recall\ \sum_j W_{ij} = 1$$

$$W_{i\bullet}^* = \arg\min_w \tfrac{1}{2} w^T C w - \lambda\left(w^T \vec{1}\right)$$

1) Take Deriv & Set to 0 $\quad Cw - \lambda\left(\vec{1}\right) = 0$

2) Solve Linear system $\quad C\left(\frac{w}{\lambda}\right) = \vec{1}$

3) Find λ $\quad w^T\vec{1} = 1$

4) Find w $\quad \lambda\left(\frac{w}{\lambda}\right)^T \vec{1} = 1$

LLE

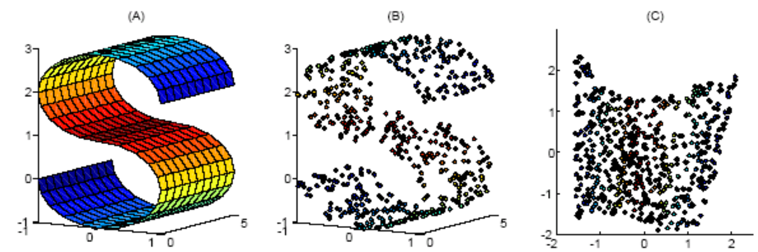
- Finding Y's (new low-D points that agree with the W's)

$$\begin{aligned}
\Phi\left(Y\right) &= \sum_i \left\|\vec{y}_i - \sum_j W_{ij}\vec{y}_j\right\|_T^2 \; subject\,to \sum_{i=1}^N \vec{y}_i = 0, \; \sum_{i=1}^N \vec{y}_i\vec{y}_i^T = I \\
&= \sum_i \left(\vec{y}_i - \sum_j W_{ij}\vec{y}_j\right)^T \left(\vec{y}_i - \sum_k W_{ik}\vec{y}_k\right) \\
&= \sum_i \left(\vec{y}_i^T\vec{y}_i - \sum_k W_{ik}\vec{y}_i^T\vec{y}_k - \sum_j W_{ij}\vec{y}_j^T\vec{y}_i + \sum_{jk} W_{ij}W_{ik}\vec{y}_j^T\vec{y}_k\right) \\
&= \sum_{jk} \left(\delta_{jk} - W_{jk} - W_{kj} + \sum_i W_{ij}W_{ik}\right)\vec{y}_j^T\vec{y}_k \\
&= \sum_{jk} M_{jk}\vec{y}_j^T\vec{y}_k \\
&= tr\left(MYY^T\right)
\end{aligned}$$

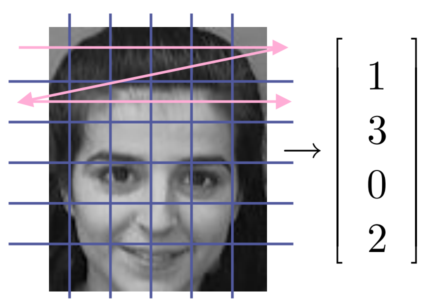
- Where Y is a matrix whose rows are the y vectors
- To minimize the above subject to constraints
  we set Y as the bottom d+1 eigenvectors of M

LLE Results

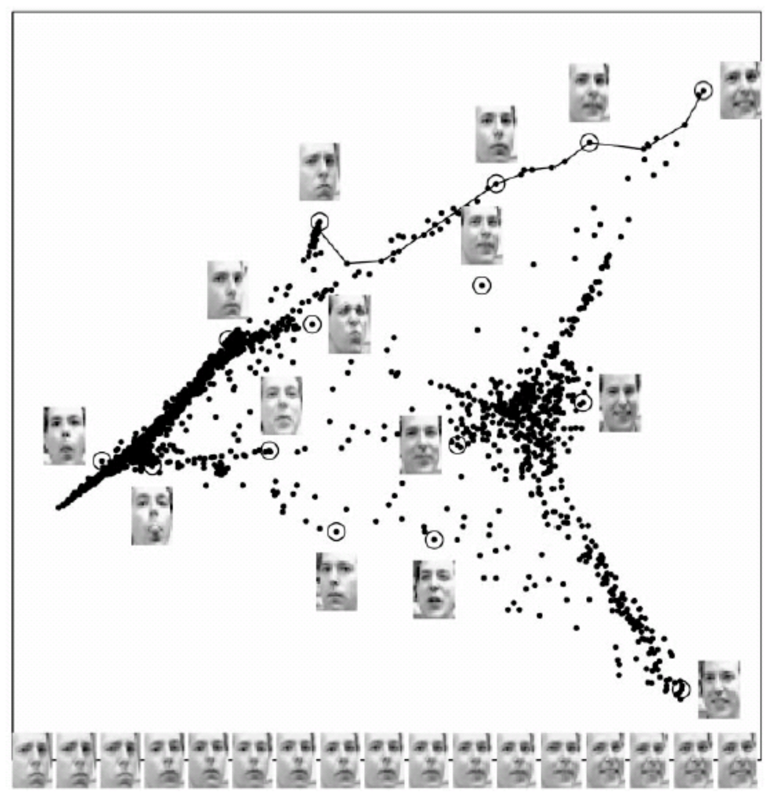
- Synthetic data on S-manifold
- Have noisy 3D samples X
- Get 2D LLE embedding Y



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

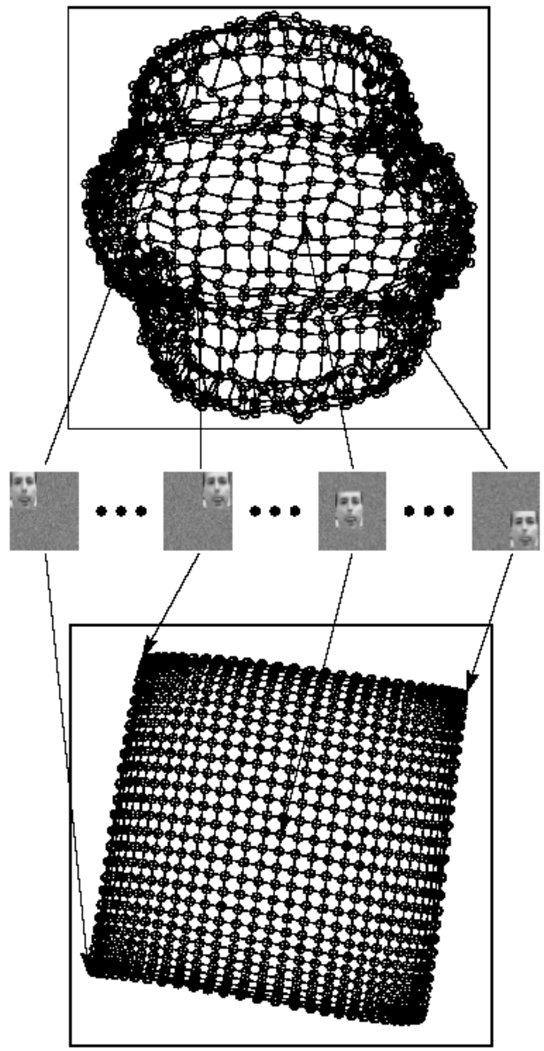


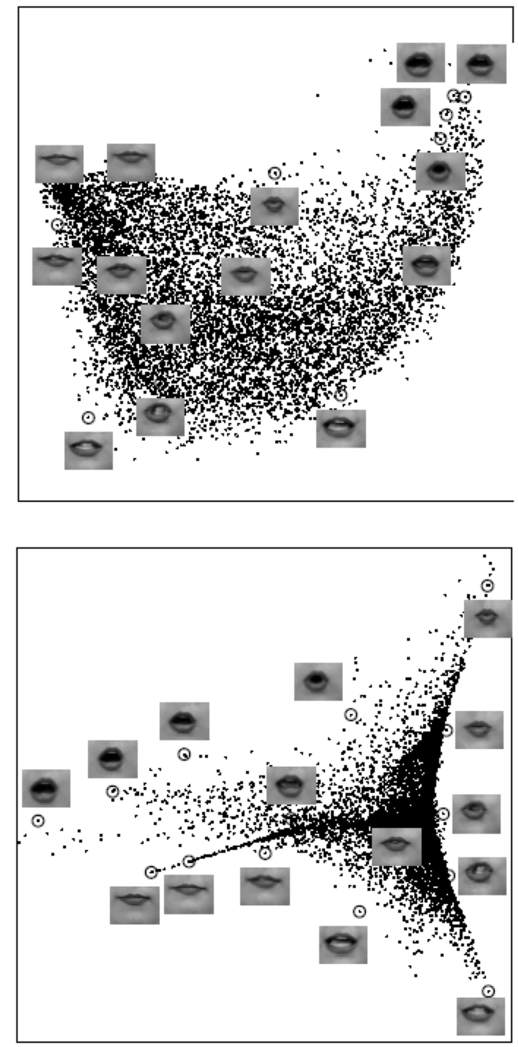
- Dots are reconstructed two-dimensional Y points



LLE Results

- Top=PCA
- Bottom=LLE





Kernel Principal Components Analysis

- Recall, PCA approximates the data with eigenvectors, mean and coefficients: $\vec{x}_i \approx \vec{\mu} + \sum_{j=1}^{C} c_{ij} \vec{v}_j$
- Get eigenvectors that best approximating the covariance:

$$\Sigma = V \Lambda V^T$$

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

- Eigenvectors are orthonormal: $\vec{v}_i^T \vec{v}_j = \delta_{ij}$
- In coordinates of v, Gaussian is diagonal, cov = Λ
- 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: $c_{ij} = \left(\vec{x}_i - \vec{\mu}\right)^T \vec{v}_j$
- 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

If data is zero-mean $C = \frac{1}{N}\sum_{i=1}^{N} \vec{x}_i \vec{x}_i^T$ $\quad \lambda \vec{v} = C\vec{v}$ **Evals & Evecs satisfy**

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

- For nonlinearity, do PCA on feature expansions:

$$\bar{C} = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T$$

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

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

- As usual, kernel must satisfy Mercer's theorem
- Assume, for simplicity, all feature data is zero-mean $\quad \sum_{i=1}^{N} \phi(x_i) = 0$

Kernel PCA

- Efficiently find & use eigenvectors of C-bar: $\lambda \vec{v} = \bar{C}\vec{v}$
- Can dot either side of above equation with feature vector:

$$\lambda \phi(x_i)^T \vec{v} = \phi(x_i)^T \bar{C}\vec{v}$$

- Eigenvectors are in span of feature vectors: $\vec{v} = \sum_{i=1}^{N} \alpha_i \phi(x_i)$
- Combine equations:

$$\lambda \phi(x_i)^T \vec{v} = \phi(x_i)^T \bar{C}\vec{v}$$

$$\lambda \phi(x_i)^T \left\{\sum_{j=1}^{N} \alpha_j \phi(x_j)\right\} = \phi(x_i)^T \bar{C} \left\{\sum_{i=1}^{N} \alpha_j \phi(x_j)\right\}$$

$$\lambda \phi(x_i)^T \left\{\sum_{j=1}^{N} \alpha_j \phi(x_j)\right\} = \phi(x_i)^T \left\{\frac{1}{N}\sum_{k=1}^{N} \phi(x_k)\phi(x_k)^T\right\} \left\{\sum_{j=1}^{N} \alpha_j \phi(x_j)\right\}$$

$$\lambda \sum_{j=1}^{N} \alpha_j K_{ij} = \frac{1}{N}\sum_{k=1}^{N} K_{ik} \sum_{j=1}^{N} \alpha_j K_{kj}$$

$$N\lambda K\vec{\alpha} = K^2\vec{\alpha}$$

$$N\lambda \vec{\alpha} = K\vec{\alpha}$$

Kernel PCA

- From before, we had: $\lambda\phi\left(x_i\right)^T \vec{v} = \phi\left(x_i\right)^T \bar{C}\vec{v}$

  this is an eig equation! $N\lambda\vec{\alpha} = K\vec{\alpha}$
- Get eigenvectors α and eigenvalues of K
- Eigenvalues are N times λ
- For each eigenvector α^k there is an eigenvector v^k
- Want eigenvectors v to be normalized: $\left(\vec{v}^k\right)^T \vec{v}^k = 1$

$$\left(\sum_{i=1}^N \alpha_i^k \phi\left(x_i\right)\right)^T \left(\sum_{j=1}^N \alpha_j^k \phi\left(x_j\right)\right) = 1$$

$$\left(\vec{\alpha}^k\right)^T K\vec{\alpha}^k = 1$$

$$\left(\vec{\alpha}^k\right)^T N\lambda^k\vec{\alpha}^k = 1$$

$$\left(\vec{\alpha}^k\right)^T \vec{\alpha}^k = \frac{1}{N\lambda^k}$$

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

Kernel PCA

- To compute k'th projection coefficient of a new point $\phi(x)$

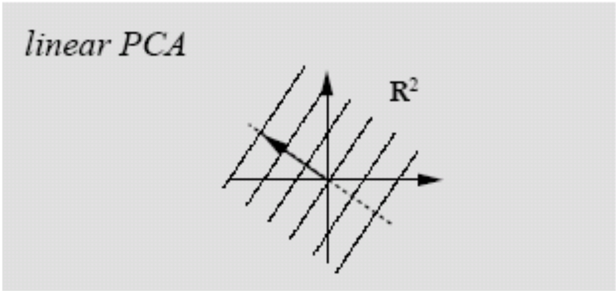
$$c^k = \phi(x)^T \vec{v}^k = \phi(x)^T \left\{ \sum_{i=1}^N \alpha_i^k \phi(x_i) \right\} = \sum_{i=1}^N \alpha_i^k k(x, x_i)$$

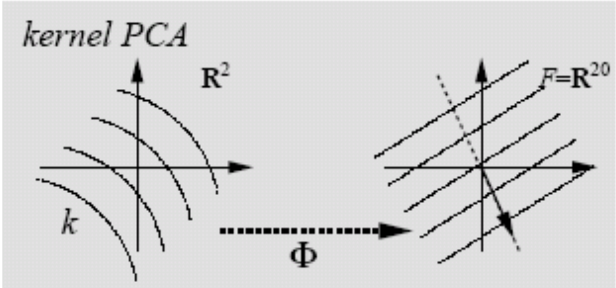
- Reconstruction*:

$$\tilde{\phi}(x) = \sum_{k=1}^K c^k \vec{v}^k = \sum_{k=1}^K \sum_{i=1}^N \alpha_i^k k(x, x_i) \sum_{j=1}^N \alpha_j^k \phi(x_j)$$

  *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…

$$
\begin{aligned}
\tilde{K}_{ij} &= \tilde{\phi}(x_i)^T \tilde{\phi}(x_j) \\
&= \left(\phi(x_i) - \tfrac{1}{N}\sum_{k=1}^{N}\phi(x_k)\right)^T \left(\phi(x_j) - \tfrac{1}{N}\sum_{k=1}^{N}\phi(x_k)\right) \\
&= \phi(x_i)^T \phi(x_j) - \tfrac{1}{N}\sum_{k=1}^{N}\phi(x_k)^T\phi(x_j) \\
&\quad - \tfrac{1}{N}\sum_{k=1}^{N}\phi(x_i)^T\phi(x_k) + \tfrac{1}{N}\tfrac{1}{N}\sum_{k=1}^{N}\sum_{l=1}^{N}\phi(x_k)^T\phi(x_l) \\
&= K_{ij} - \tfrac{1}{N}\sum_{k=1}^{N}K_{kj} - \tfrac{1}{N}\sum_{k=1}^{N}K_{ik} + \tfrac{1}{N^2}\sum_{k=1}^{N}\sum_{l=1}^{N}K_{kl}
\end{aligned}
$$

- 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

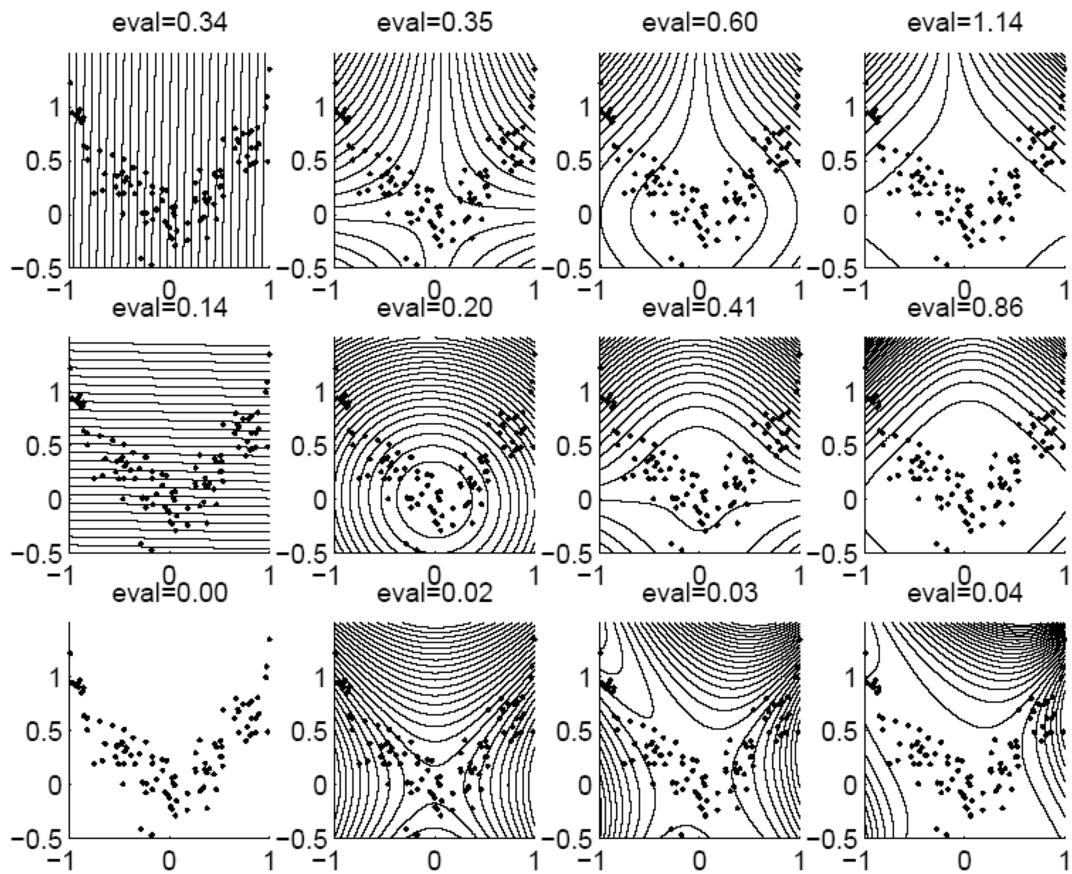
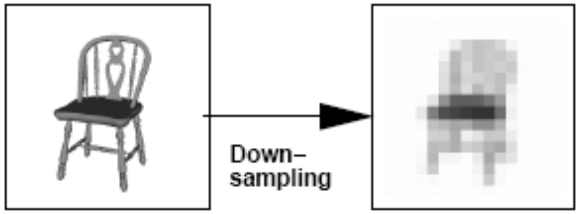


Figure 2: Two–dimensional toy examples, with data generated in the following way: x–values have uniform distribution in $[-1, 1]$, y–values are generated from $y_i = x_i^2 + \xi$, were ξ is normal noise with standard deviation 0.2. From left to right, the polynomial degree in the kernel (22) increases 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=linear as in regular PCA



# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
64	23.0	21.0	17.6	16.8	16.5	16.7	16.6
128	17.6	9.9	7.9	7.1	6.2	6.0	5.8
256	16.8	6.0	4.4	3.8	3.4	3.2	3.3
512	n.a.	4.4	3.6	3.9	2.8	2.8	2.6
1024	n.a.	4.1	3.0	2.8	2.6	2.6	2.4
2048	n.a.	4.1	2.9	2.6	2.5	2.4	2.2

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

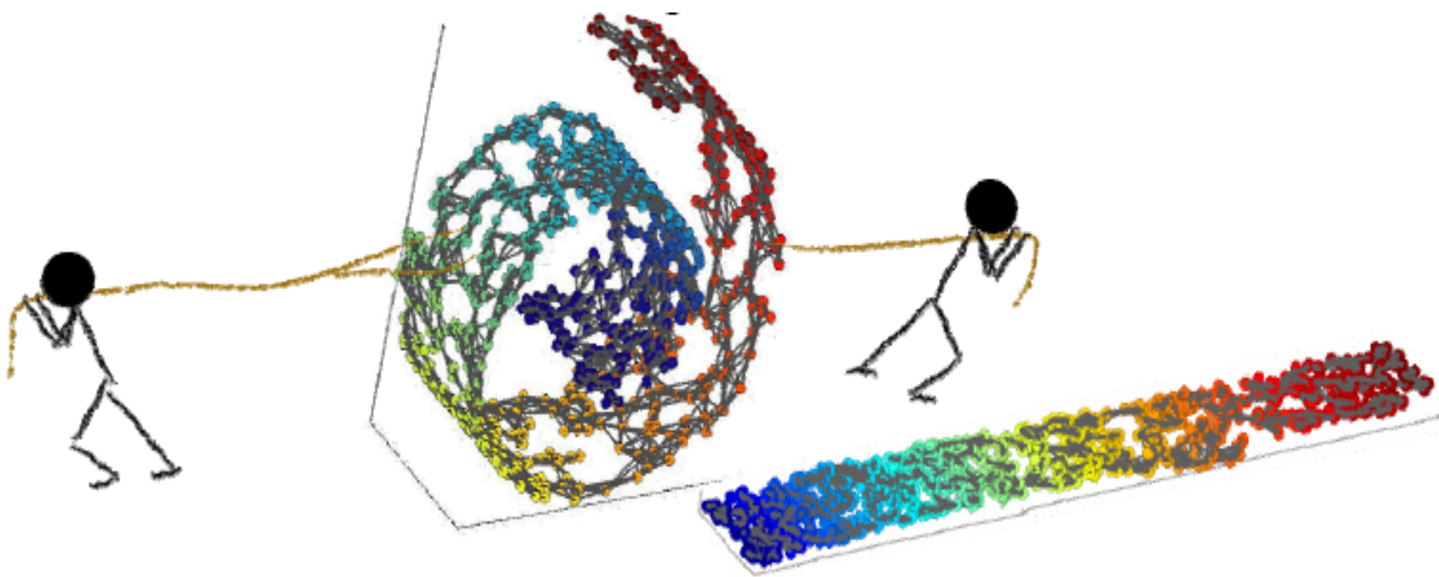
- Use coefficients of the KPCA for training a linear SVM classifier to recognize characters from their images.
- Use various polynomial kernel degrees where 1=linear as in regular PCA (worst case in experiments)
- Inferior performance to nonlinear SVMs (why??)

# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
32	9.6	8.8	8.1	8.5	9.1	9.3	10.8
64	8.8	7.3	6.8	6.7	6.7	7.2	7.5
128	8.6	5.8	5.9	6.1	5.8	6.0	6.8
256	8.7	5.5	5.3	5.2	5.2	5.4	5.4
512	n.a.	4.9	4.6	4.4	5.1	4.6	4.9
1024	n.a.	4.9	4.3	4.4	4.6	4.8	4.6
2048	n.a.	4.9	4.2	4.1	4.0	4.3	4.4

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:



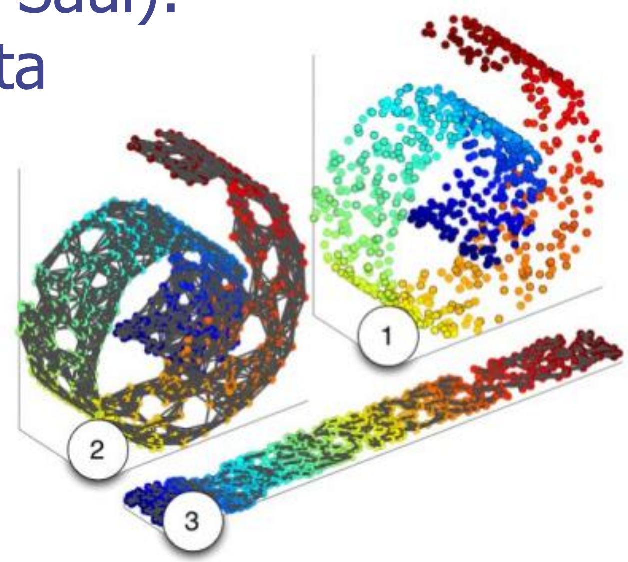
- Then apply PCA (or kPCA)

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

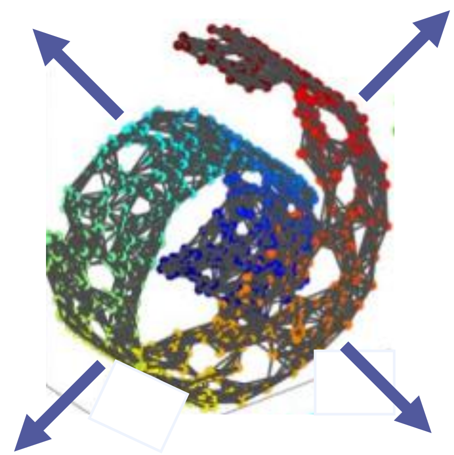


Semidefinite Embedding

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

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

$$\begin{aligned}
\kappa &= \forall K \in \Re^{N\times N} \\
& s.t. K \succeq 0 \\
& s.t. \sum_{ij} K_{ij} = 0 \\
& s.t. K_{ii} + K_{jj} - K_{ij} - K_{ji} = \\
& \qquad A_{ii} + A_{jj} - A_{ij} - A_{ji} \quad if\ C_{ij} = 1
\end{aligned}$$



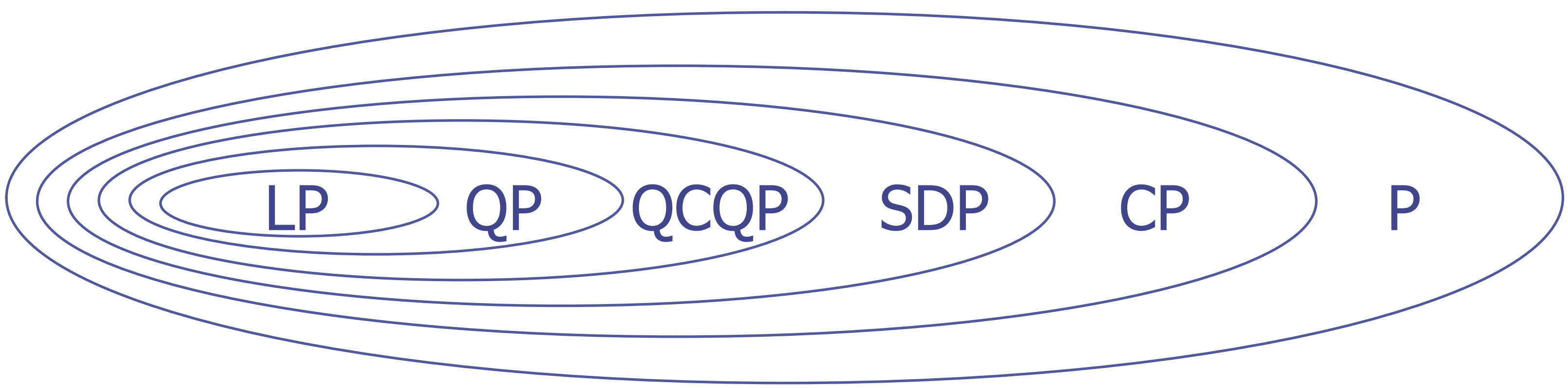
- SDE's stretching of graph *improves* the visualization

A

K

SDE Optimization with YALMIP

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



SDE Optimization with YALMIP

- LP $\min_{\vec{x}} \vec{b}^T \vec{x} \;\; s.t. \; \vec{c}_i^T \vec{x} \geq \alpha_i \;\; \forall i$ fast O(N^3)
- QP $\min_{\vec{x}} \frac{1}{2} \vec{x}^T H \vec{x} + \vec{b}^T \vec{x} \;\; s.t. \; \vec{c}_i^T \vec{x} \geq \alpha_i \;\; \forall i$
- QCQP $\min_{\vec{x}} \frac{1}{2} \vec{x}^T H \vec{x} + \vec{b}^T \vec{x} \;\; s.t. \; \vec{c}_i^T \vec{x} \geq \alpha_i \;\; \forall i, \;\; \vec{x}^T \vec{x} \leq \eta$
- SDP $\min_K tr\left(BK\right) \;\; s.t. \; tr\left(C_i^T K\right) \geq \alpha_i \;\; \forall i, \;\; K \succeq 0$

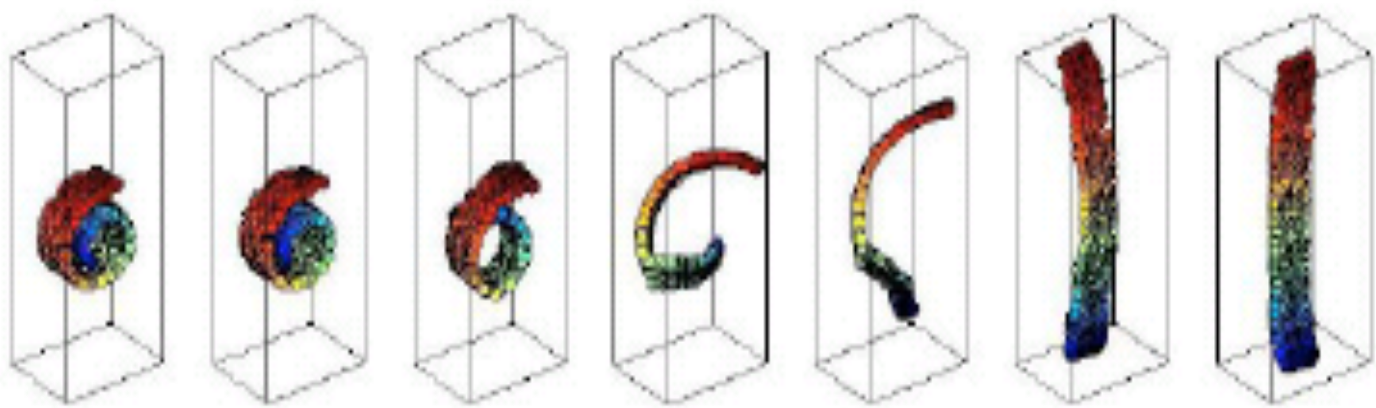
… ALL above in the YALMIP package for Matab!

… Google it, download and install!

- CP $\min_{\vec{x}} f\left(\vec{x}\right) \;\; s.t. \; g\left(\vec{x}\right) \geq \alpha$ slow O(N^3)

SDE Results

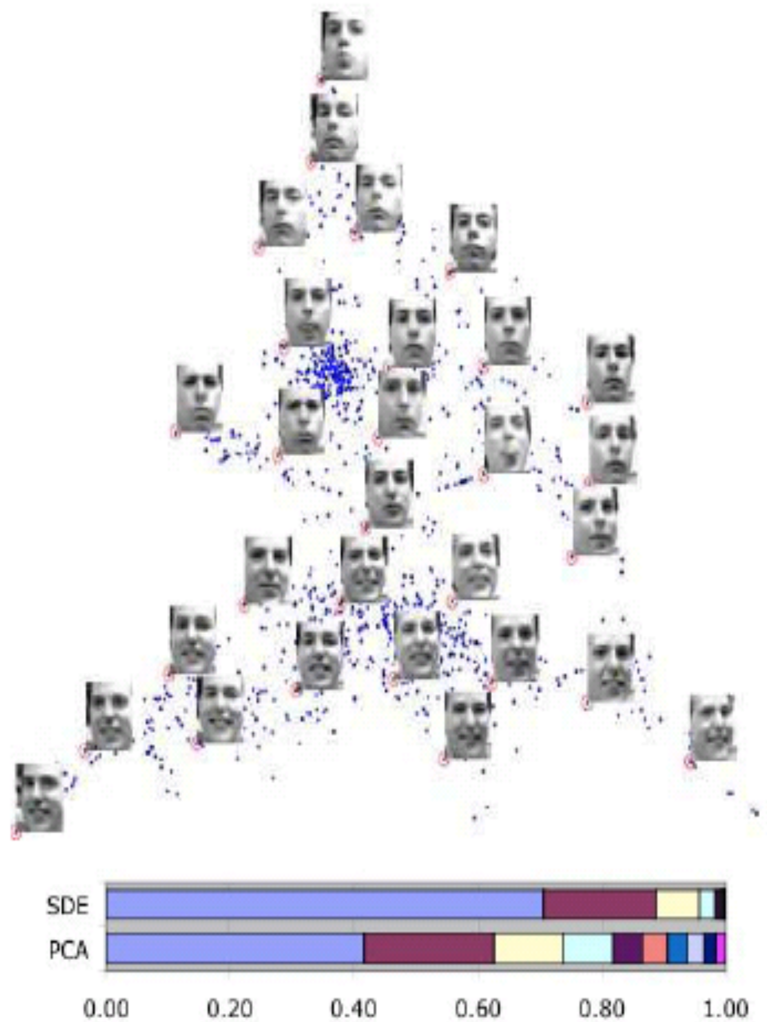
- SDE unfolds (pulls apart) knn connected graph C



- SDE's stretching of graph *improves* the visualization here

SDE Results

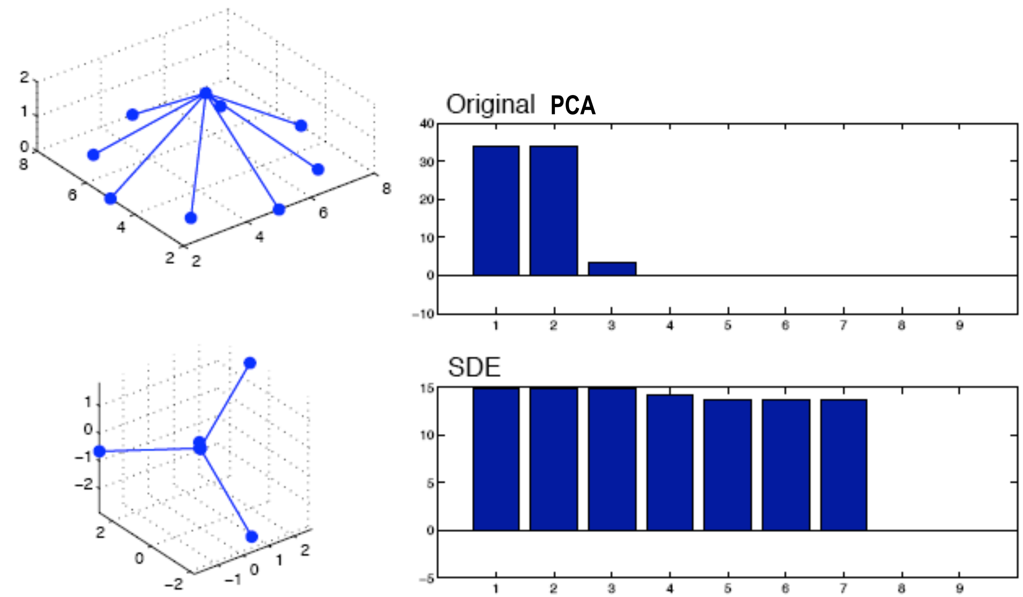
- SDE unfolds (pulls apart) knn connected graph C before doing PCA
- Gets more use or energy out of the top eigenvectors than PCA
- SDE's stretching of graph *improves* visualization here



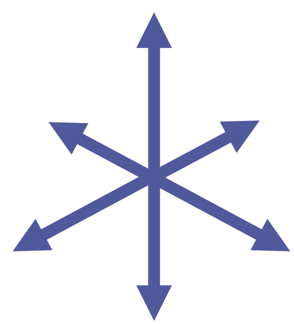
SDE Problems → MVE

- But SDE stretching could *worsen* visualization!

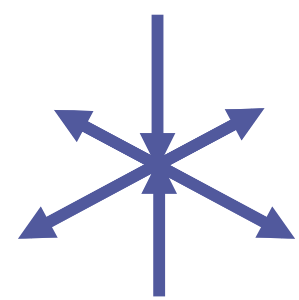
- Spokes Experiment:



- Want to pull apart only in visualized dimensions
- Flatten down remaining ones

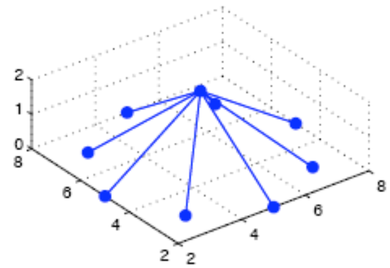


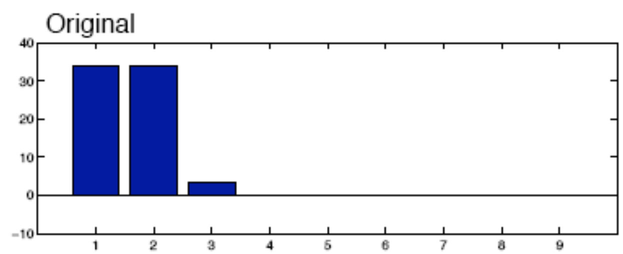
vs.



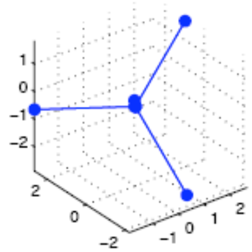
Minimum Volume Embedding

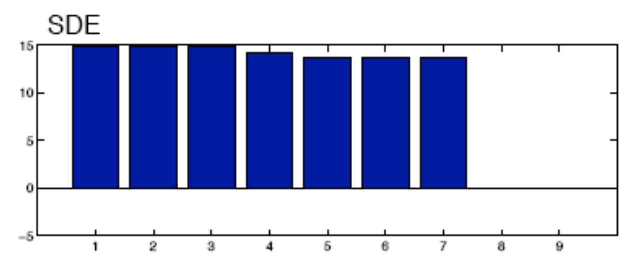
- To reduce dimension, drive energy into top (e.g. 2) dims
- Maximize fidelity $F(K) = \dfrac{\lambda_1 + \lambda_2}{\sum_i \lambda_i}$ or % energy in top dims





$$F(K) = 0.98$$





$$F(K) = 0.29$$

- Equivalent to maximizing $\lambda_1 + \lambda_2 - \beta \sum_i \lambda_i$ for some β
- Assume β=1/2…

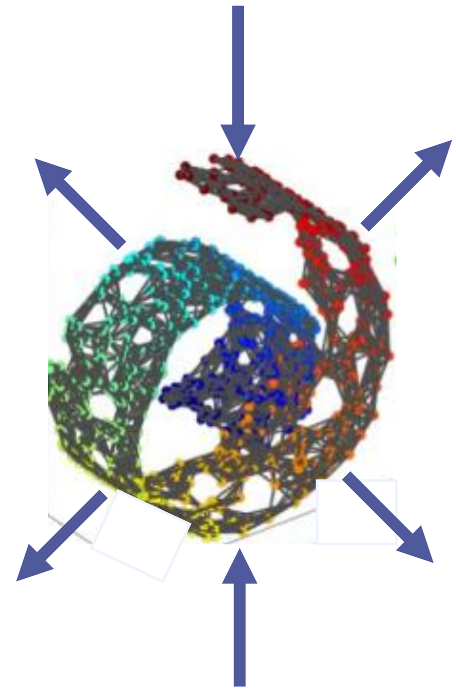
Minimum Volume Embedding

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

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

- Simplest Linear-Spectral SDP…

$$\vec{\alpha} = \begin{bmatrix} \alpha_1 & \cdots & \alpha_d & \alpha_{d+1} & \cdots & \alpha_D \end{bmatrix}$$
$$= \begin{bmatrix} +1 & \cdots & +1 & -1 & \cdots & -1 \end{bmatrix}$$



- Effectively maximizes Eigengap between d'th and d+1'th λ

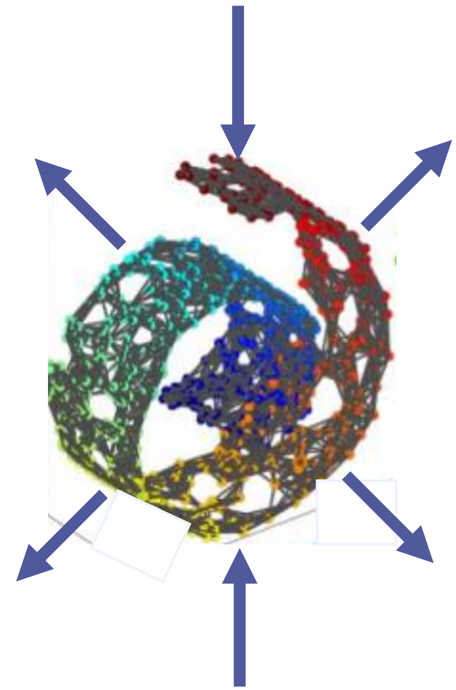
Minimum Volume Embedding

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

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

- Simplest Linear-Spectral SDP…

$$\vec{\alpha} = \begin{bmatrix} \alpha_1 & \cdots & \alpha_d & \alpha_{d+1} & \cdots & \alpha_D \end{bmatrix}$$
$$= \begin{bmatrix} +1 & \cdots & +1 & -1 & \cdots & -1 \end{bmatrix}$$



- Effectively maximizes Eigengap between d'th and d+1'th λ

- Variational bound on cost → 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 $\max_{K\in\kappa} tr\left(BK\right)$

Logdet SDPs $\max_{K\in\kappa} \sum_i \log \lambda_i$

- Consider richer SDPs (assume λ_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 $g(K) = \sum_i \alpha_i \lambda_i$
  get a variational SDP problem (a Procrustes problem).

$$
\begin{aligned}
\max_{K \in \kappa} g(K) &= \max_{K \in \kappa} \sum_i \alpha_i \lambda_i \\
&= \max_{K \in \kappa} \sum_i \alpha_i \lambda_i tr\left(v_i^T v_i\right) \\
&= \max_{K \in \kappa} \sum_i \alpha_i tr\left(\lambda_i v_i v_i^T\right) \\
&= \max_{K \in \kappa} \sum_i \alpha_i tr\left(K v_i v_i^T\right) \\
&= \max_{K \in \kappa} tr\left(K \sum_i \alpha_i v_i v_i^T\right) \\
&= \begin{cases} \max_{K \in \kappa} \min_U tr\left(K \sum_i \alpha_i u_i u_i^T\right) \; s.t. \; u_i^T u_j = \delta_{ij} & if \; \alpha_i \leq \alpha_{i+1} \\ \max_{K \in \kappa} \max_U tr\left(K \sum_i \alpha_i u_i u_i^T\right) \; s.t. \; u_i^T u_j = \delta_{ij} & if \; \alpha_i \geq \alpha_{i+1} \end{cases}
\end{aligned}
$$

$$
s.t. \; K v_i = \lambda_i v_i \qquad \lambda_i \geq \lambda_{i+1} \qquad v_i^T v_j = \delta_{ij}
$$

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

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

$$f_d(K) = \sum_{i=1}^{d} \lambda_i \Rightarrow \; convex$$

Our linear-spectral cost is a combination of these

$$\begin{aligned} g(K) &= \alpha_D f_D(K) + \sum_{i=D-1}^{1} (\alpha_i - \alpha_{i+1}) f_i(K) \\ &= \alpha_D tr(K) + \sum_{i=D-1}^{1} \left|\alpha_i - \alpha_{i+1}\right| f_i(K) \end{aligned}$$

Trace (linear) + conic combo of convex fn's =convex

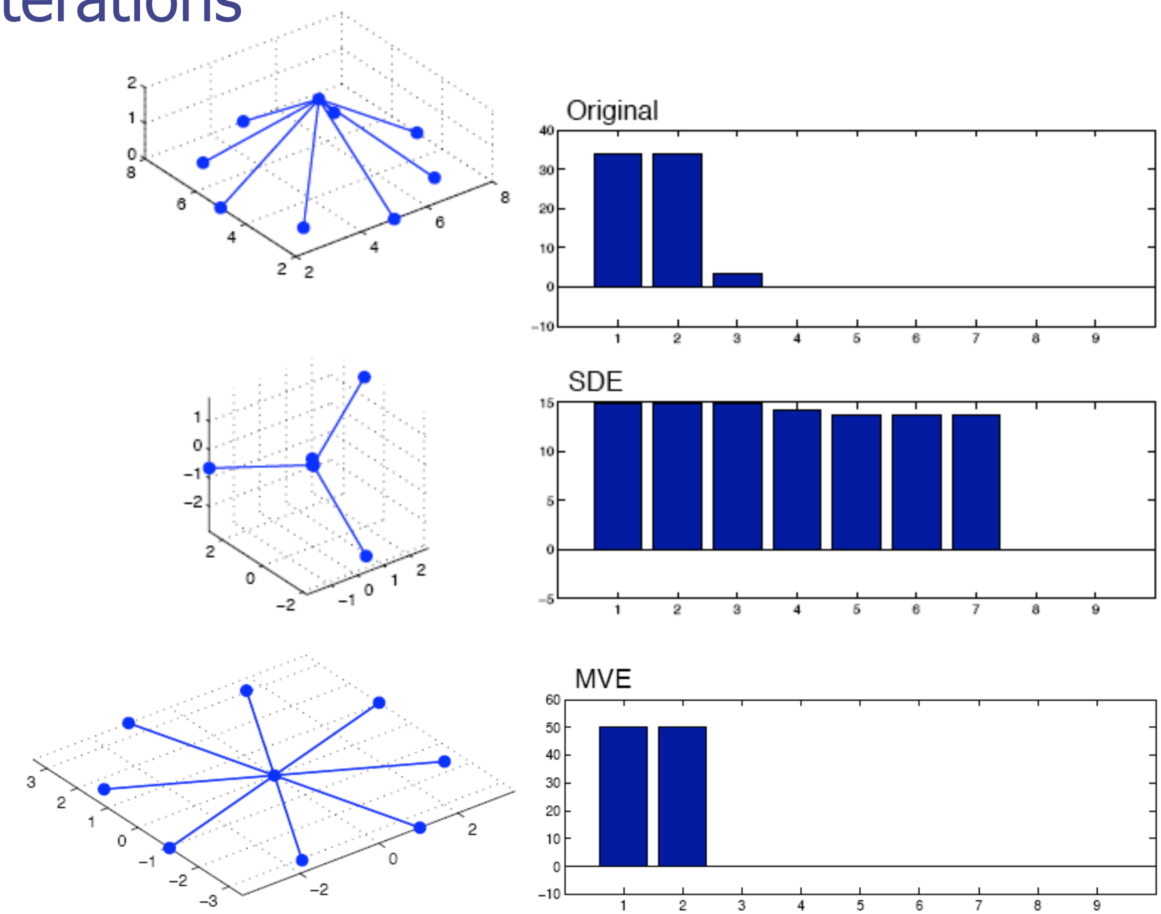
MVE Pseudocode

- Download at www.metablake.com/mve

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

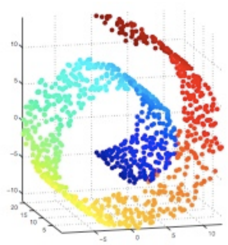
MVE Results

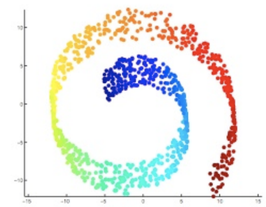
- Spokes experiment visualization and spectra
- Converges in ~5 iterations



MVE Results

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



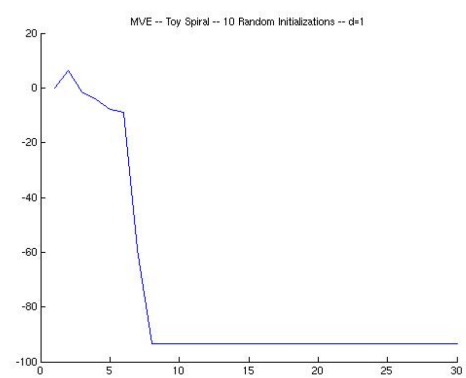


PCA

SDE

MVE

- Same convergence
  under random initialization
  or K=A…

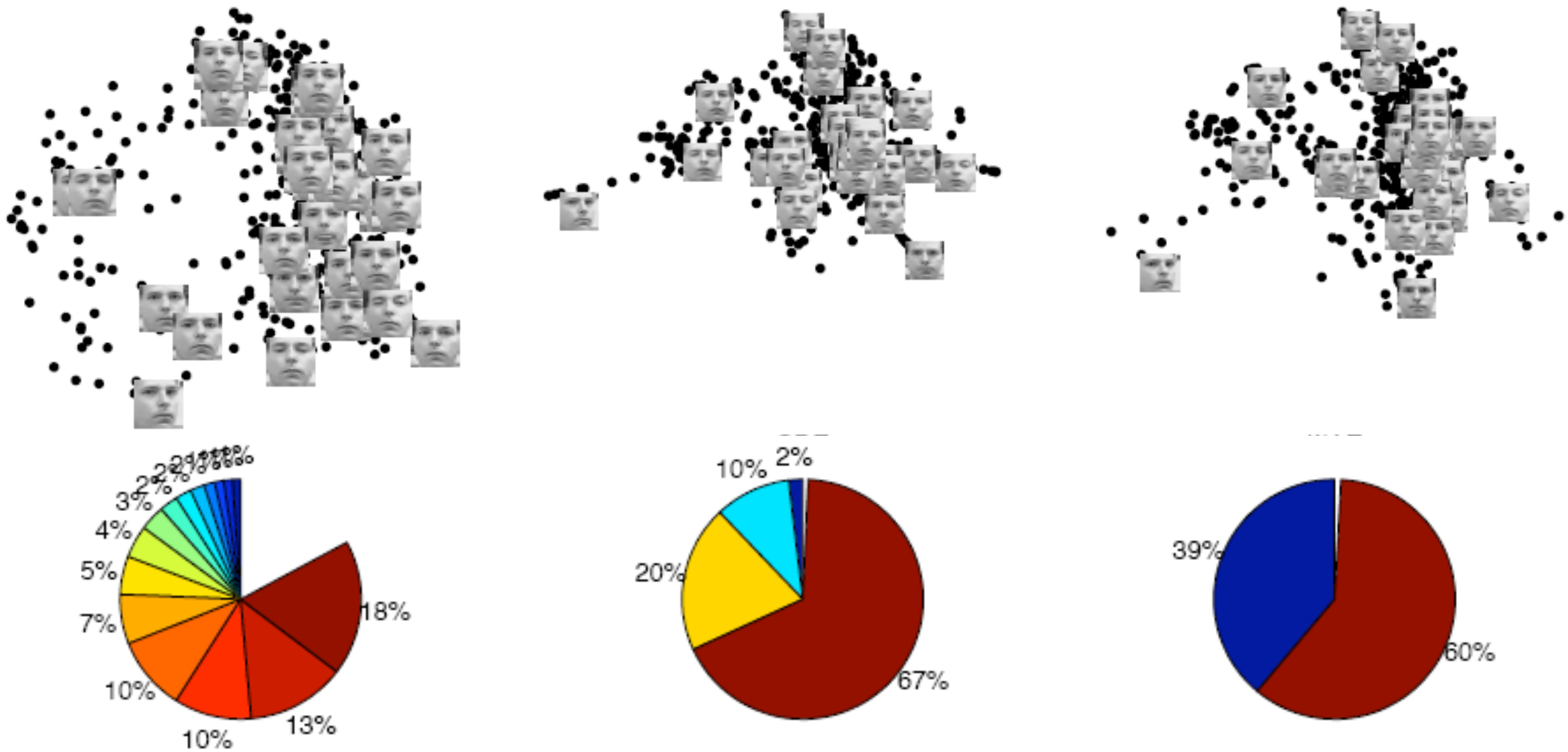


Tony Jebara

kNN Embedding with MVE

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

PCA SDE MVE

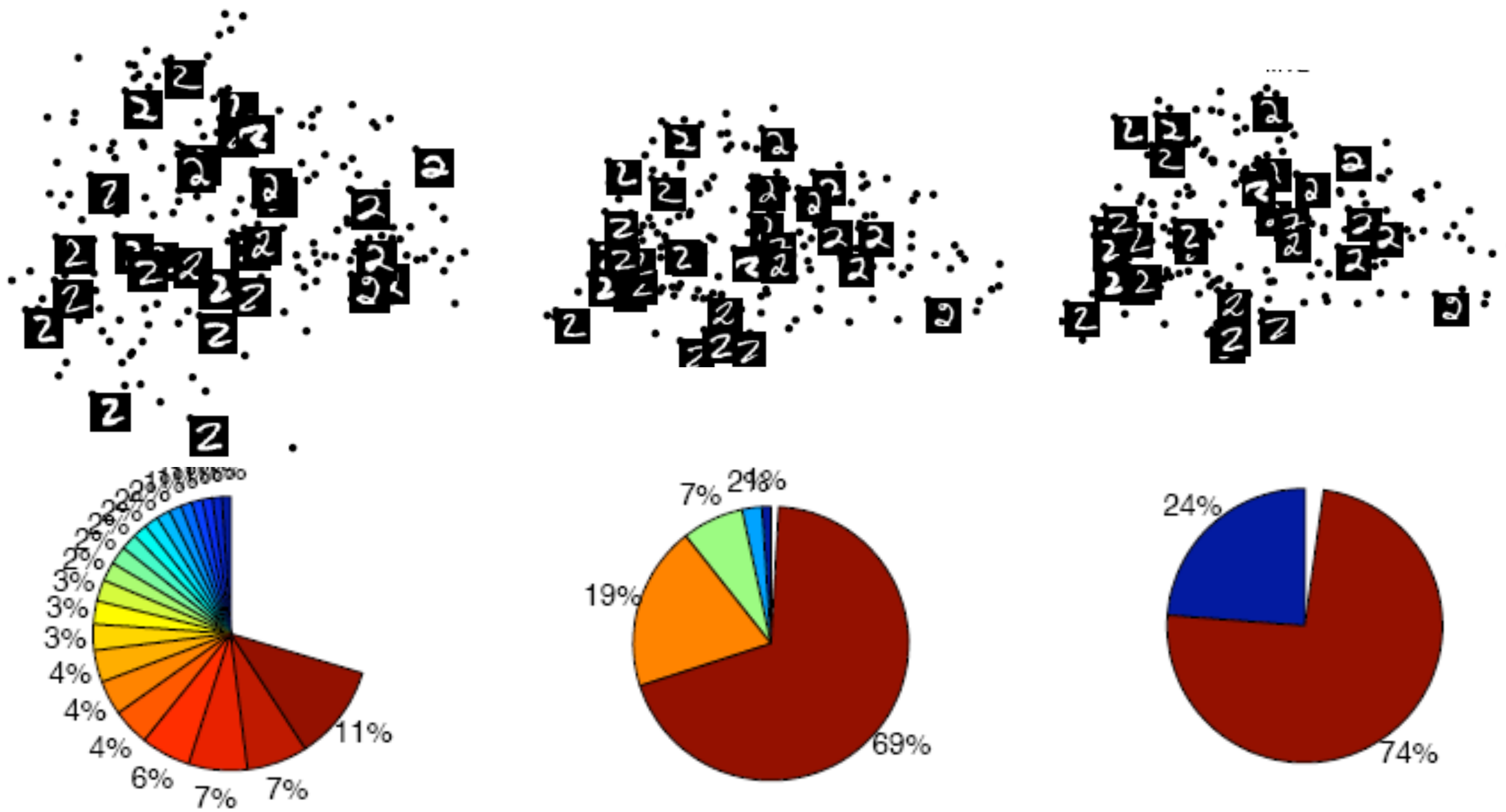


Tony Jebara

kNN Embedding with MVE

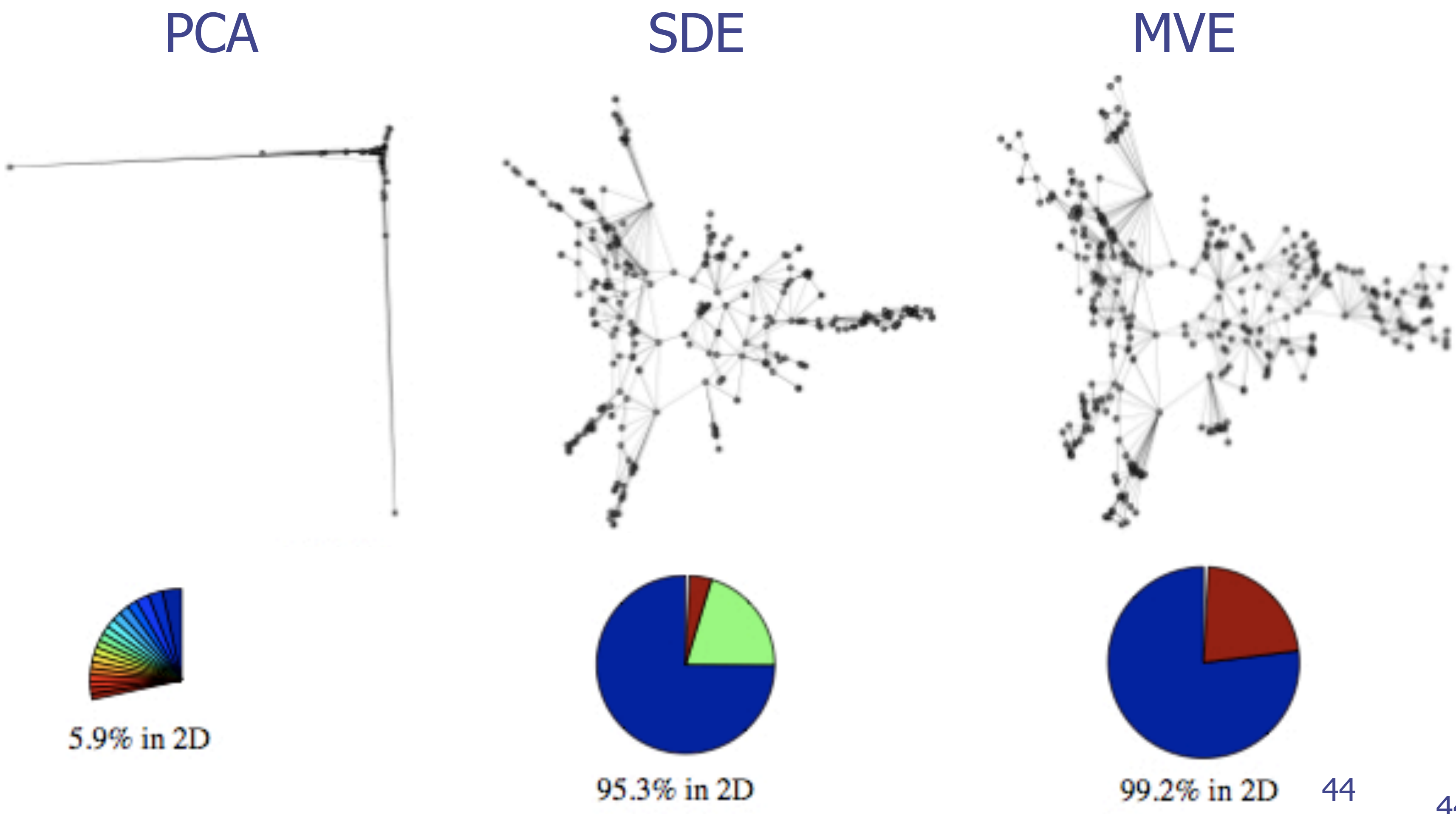
- MVE does better even with kNN
- Digit images visualization with spectra

PCA SDE MVE



Graph Embedding with MVE

- Collaboration network with spectra



44

MVE Results

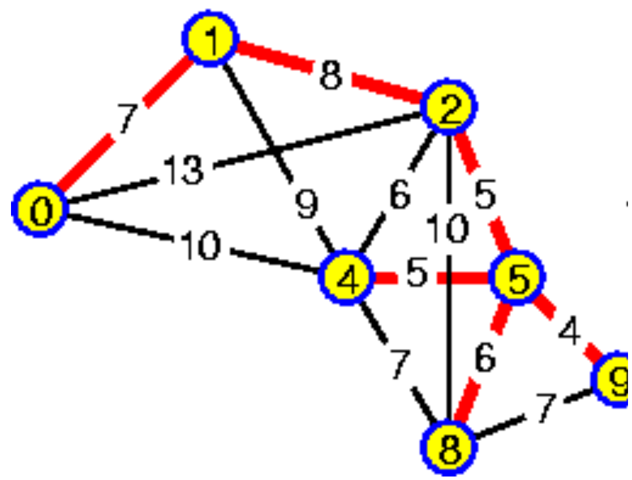
- Evaluate fidelity or % energy in top 2 dims

	kPCA	SDE	MVE
Star	95.0%	29.9%	100.0%
Swiss Roll	45.8%	99.9%	99.9%
Twos	18.4%	88.4%	97.8%
Faces	31.4%	83.6%	99.2%
Network	5.9%	95.3%	99.2%

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)

$$K \in original\,\kappa$$

$$and\ K_{ii} + K_{jj} - K_{ij} - K_{ji} \geq A_{ii} + A_{jj} - A_{ij} - A_{ji}$$

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