CSCI 4360/6360 Data Science II

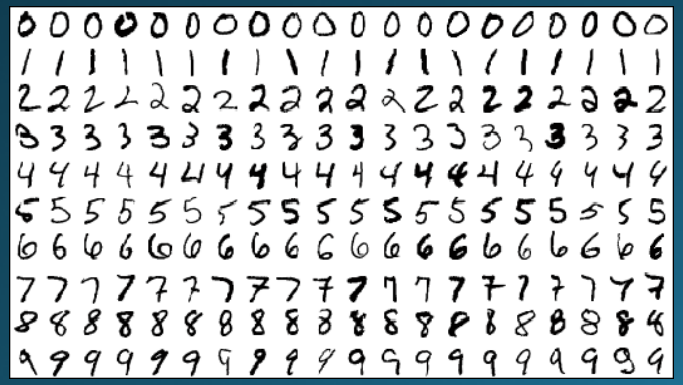
Kernel & Sparse PCA

Embeddings

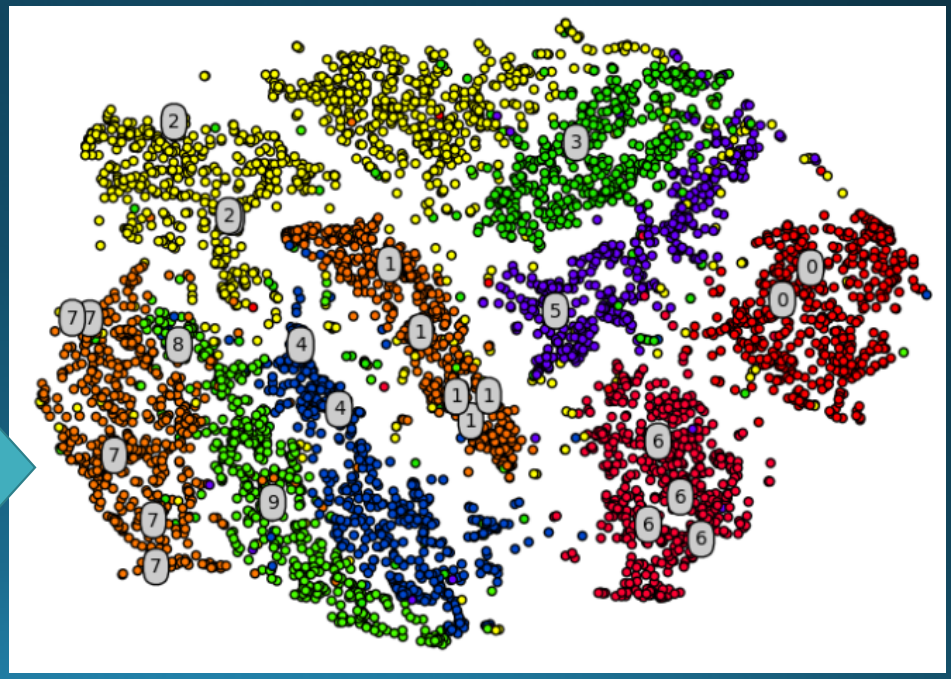
- What is an *embedding*?

$$f : X \rightarrow Y$$

- **Mapping**
- **Transformation**
- **Reveals / preserves "structure"**

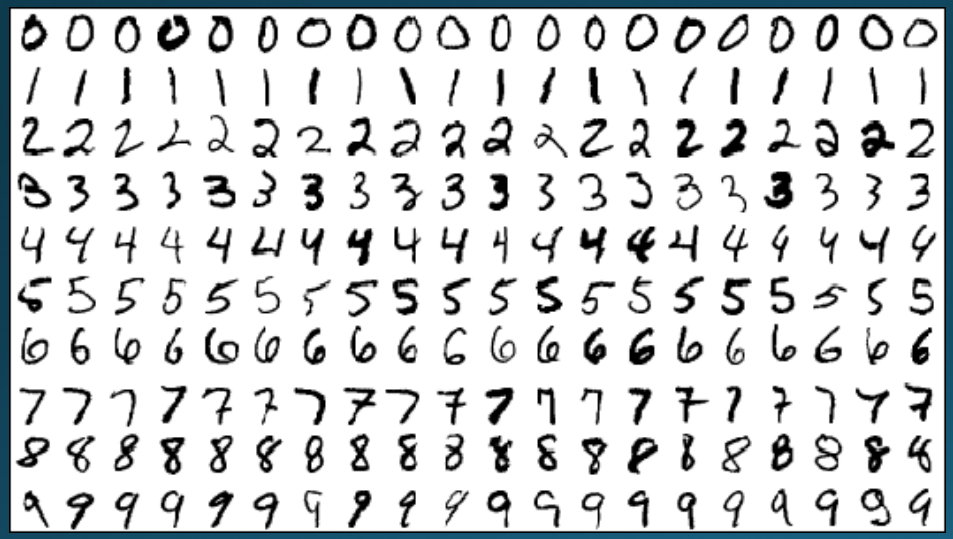


Embedding



Embeddings

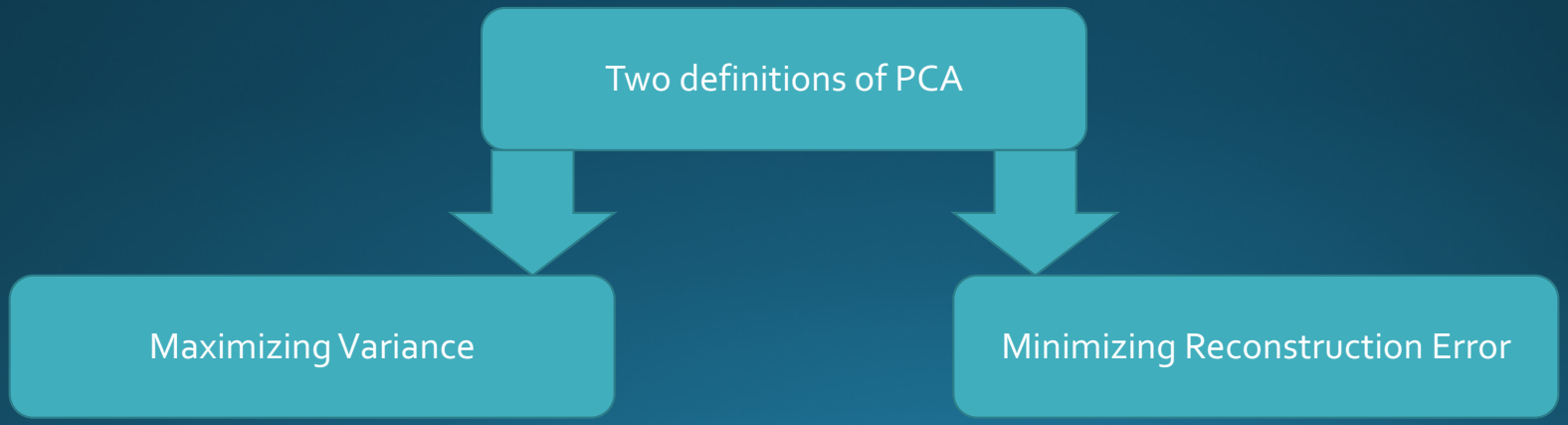
- "Degrees of freedom" versus "intrinsic dimensionality"



- Despite 64x64 pixels, only so many ways to draw a 9
- **Low-dimensional manifold**

Principal Component Analysis (PCA)

1. Orthogonal projection of data
2. Lower-dimensional linear space known as the *principal subspace*
3. Variance of the projected data is maximized



Maximizing Variance

- We start with the idea of projection from *D*-dimensions *x* to *M*-dimensions *u*
  - *u* is a unit vector, so $u^T u = 1$.
- Mean of projected data is $u^T \bar{x}$, where $\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$
- Variance of the projected data $\frac{1}{N}\sum_{n=1}^{N}\{\vec{u}_1^T\vec{x}_n - \vec{u}_1^T\bar{x}\}^2 = \vec{u}_1^T S\vec{u}_1$
- where S is the sample covariance matrix of the data $S = \frac{1}{N}\sum_{n=1}^{N}(\vec{x}_n - \bar{x})(\vec{x}_n - \bar{x})^T$

Maximizing Variance

- We want to maximize projected variance $u_1^T S u_1$ with respect to u_1
- Obvious problem: needs to be constrained, or else $||u_1||$ -> ∞

- Appropriate constraint: $u_1^T u_1 = 1$, enforced with Lagrange multiplier

$$\vec{u}_1^T S \vec{u}_1 + \lambda_1 (1 - \vec{u}_1^T \vec{u}_1)$$

- Set derivative with respect to $u_1 = 0$, and a stationary point appears

$$S\vec{u}_1 = \lambda_1 \vec{u}_1$$

- Means u_1 must be an eigenvector of S! Left-multiply by u_1^T

$$\vec{u}_1^T S \vec{u}_1 = \lambda_1$$

- Variance will be max when these are 1st eigenvalue & eigenvector

Minimizing Error

- We want the reconstruction error using the first $M < D$ principal components to be minimal

$$J = \frac{1}{N} \sum_{n=1}^{N} ||\vec{x}_n - \tilde{x}_n||^2$$

We want to minimize J

- This can be rewritten purely in terms of eigenvectors u_i

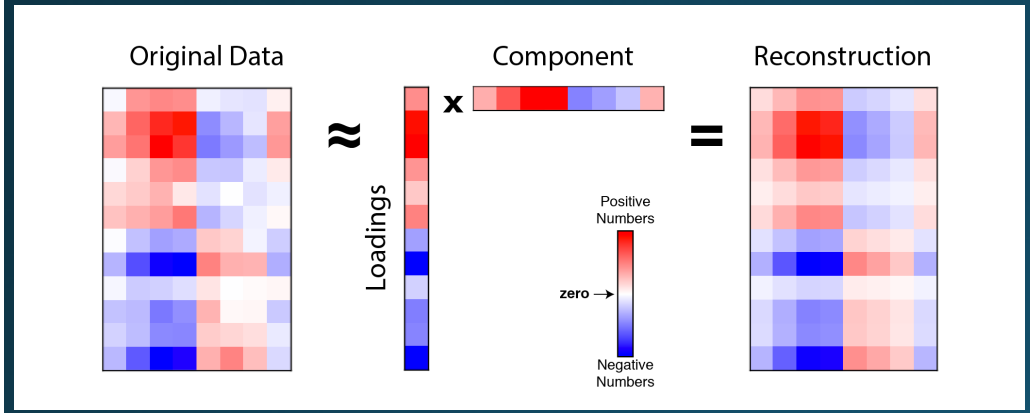
$$J = \sum_{n=M+1}^{D} \vec{u}_i^T S \vec{u}_i$$

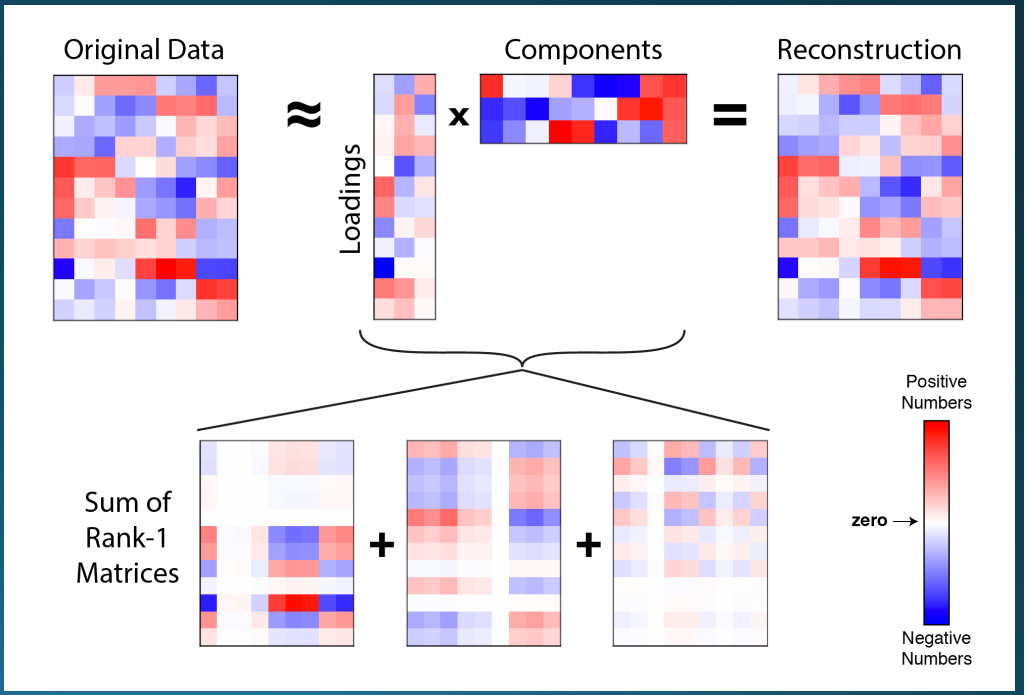
Eigenvectors u_i come out of equation for $\tilde{x}$

- Therefore, the distortion measure of reconstruction using the M eigenvectors of the largest eigenvalues is the sum of the remaining D - M eigenvalues

$$J = \sum_{n=M+1}^{D} \lambda_i$$

Minimizing Error





Principal Component Analysis

Advantages

- Optimal low-rank approximation in terms of squared reconstruction error
- Completely unsupervised
- Endless applications

Disadvantages

- Principal components are linear combinations (cannot generate nonlinear PCs; struggles to determine PCs in geodesic spaces)
- Basis vectors are dense and sometimes difficult to interpret

Kernel PCA

- Whenever we compute a *kernel*, we rely on a scalar (dot) product of the form $x^T x$
- Conventional PCA is an outer product (covariance), $X^T X$
- What if we replaced this with an inner product, XX^T
  - This "Gram matrix" is what we compute eigenvectors of in PCA anyway
- If anything, Kernel PCA is a *generalization* of PCA to arbitrary similarity (kernel) functions!
- **First step**: express conventional PCA such that data vectors x appear only in the form of scalar products

Kernel PCA

- Recall that the principal components are defined by eigenvectors of the covariance matrix

$$S\vec{u}_i = \lambda_i \vec{u}_i$$

i is the dimensional index

- and sample covariance matrix defined by

$$S = \frac{1}{N} \sum_{n=1}^{N} \vec{x}_n \vec{x}_n^T$$

N is the number of data points

- and eigenvectors are normalized such that

$$\vec{u}_i^T \vec{u}_i = 1$$

Kernel PCA

- In kernel PCA, we consider data that have already undergone a nonlinear transformation:

$$\vec{x} \in \mathcal{R}^D \quad \Rightarrow \quad \phi(\vec{x}) \in \mathcal{R}^M$$

- **We now perform PCA on this new *M*-dimensional feature space**

Kernel PCA

- Sample covariance matrix *C* (now *MxM*)

$$C = \frac{1}{N}\sum_{n=1}^{N}\phi(\vec{x}_n)\phi(\vec{x}_n)^T \qquad C\vec{v}_i = \lambda_i\vec{v}_i$$

- **Goal:** solve the eigenvector/eigenvalue equation **without having to explicitly operate in the *M*-dimensional feature space**
- Combining the two equations: $\frac{1}{N}\sum_{n=1}^{N}\phi(\vec{x}_n)\phi(\vec{x}_n)^T\vec{v}_i = \lambda_i\vec{v}_i$
- This reduces to

$$\vec{v}_i = \sum_{n=1}^{N} a_{in}\phi(\vec{x}_n)$$

Kernel PCA

- Substitute back into eigenvector equation and we get a royal mess

$$\frac{1}{N}\sum_{n=1}^{N}\phi(\vec{x}_n)\phi(\vec{x}_n)^T\sum_{m=1}^{N}a_{im}\phi(\vec{x}_m)=\lambda_i\sum_{n=1}^{N}a_{in}\phi(\vec{x}_n)$$

- Remember our goal: work only in terms of $k(x_n, x_m) = \phi(x_n)^T\phi(x_m)$
- Multiply both sides by $\phi(x_l)$

$$\frac{1}{N}\sum_{n=1}^{N}k(\vec{x}_l,\vec{x}_n)\sum_{m=1}^{N}a_{im}k(\vec{x}_n,\vec{x}_m)=\lambda_i\sum_{n=1}^{N}a_{in}k(\vec{x}_l,\vec{x}_n)$$

Kernel PCA

$$\frac{1}{N}\sum_{n=1}^{N} k(\vec{x}_l, \vec{x}_n) \sum_{m=1}^{N} a_{im} k(\vec{x}_n, \vec{x}_m) = \lambda_i \sum_{n=1}^{N} a_{in} k(\vec{x}_l, \vec{x}_n)$$

- Look familiar?

$$K^2 \vec{v}_i = \lambda_i K \vec{v}_i$$

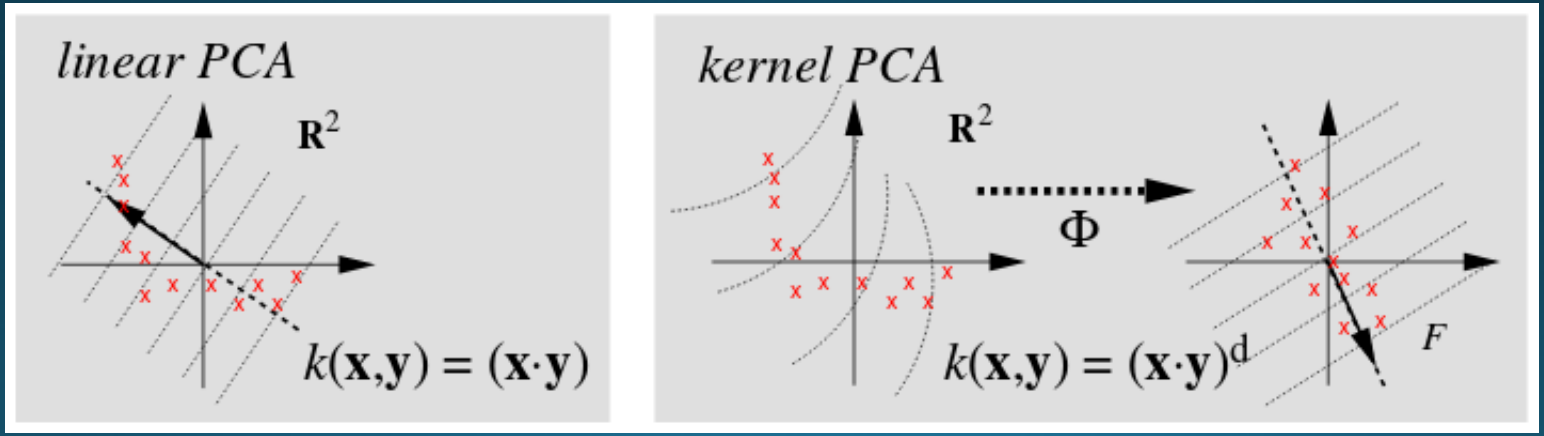
- Which reduces to

$$K\vec{v}_i = \lambda_i \vec{v}_i$$

- (there's some normalization magic that has to happen but we're skipping that for now)

Kernel PCA

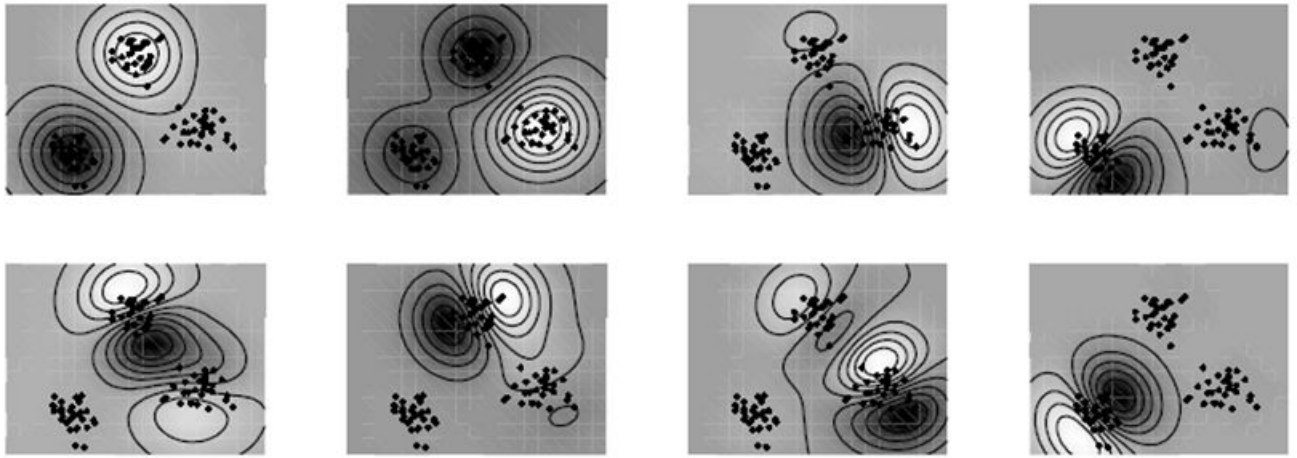
- Data in original data space (right panel, left subpanel) projected by nonlinear transformation into feature space (right subpanel). By performing PCA on feature space, PCs correspond to nonlinear projections in original data space.



Kernel PCA

- Gaussian kernel applied to 2D data
- First 8 kernel PCs
- Contours are lines along which the projection onto the corresponding PC is constant

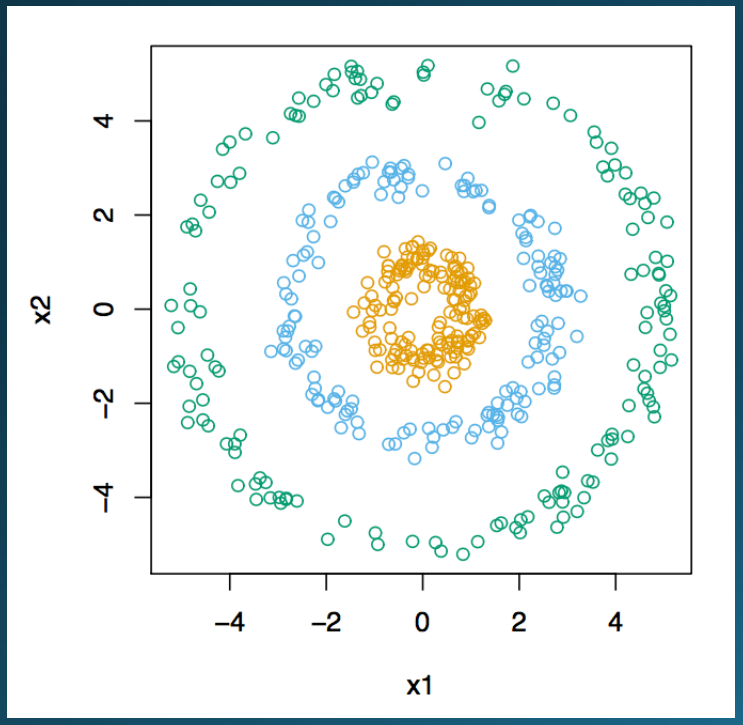
$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{y}\|^2\right)$$



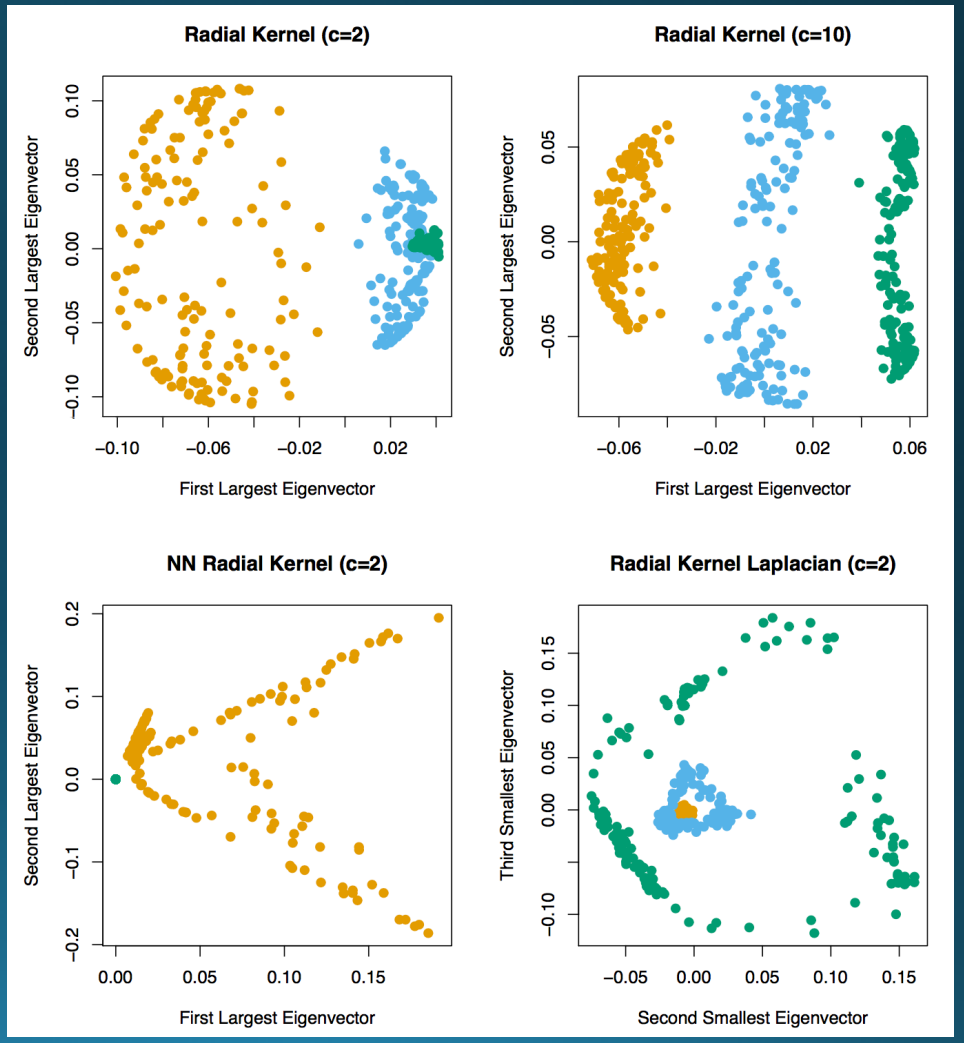
(courtesy of Bernhard Schölkopf)

Kernel PCA

Data



Kernelized PCs



Kernel PCA

Advantages

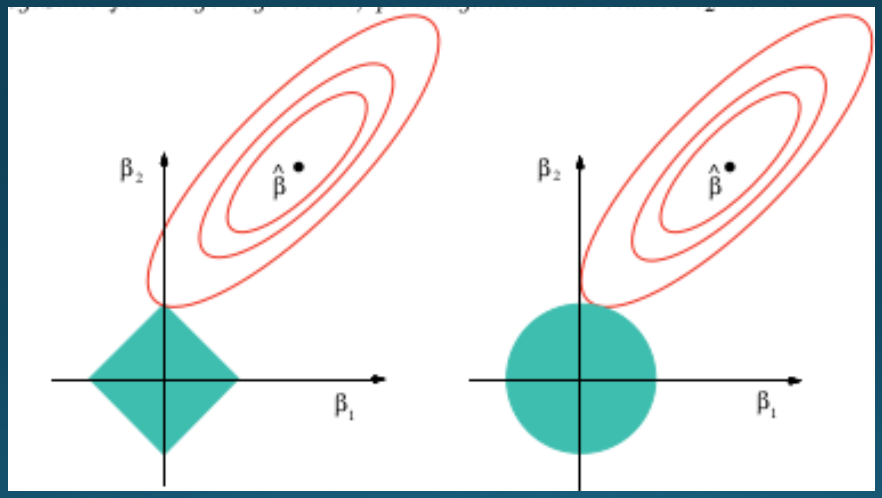
- Allows for nonlinear principal components
- Infinitely flexible in terms of allowed kernel functions

Disadvantages

- Requires finding eigenvectors and eigenvalues of *NxN* matrix, instead of *DxD* (large *N* is problematic)
- Cannot project new, unobserved data onto *L*-dimensional manifold of kernel

Sparse PCA

- Anyone remember lasso regularization?



- Regularization, in general, is a penalty to encourage small weights (remember Assignment 2)
- Lasso (or L_1) forces weights to 0 so they become *sparse*

Sparse PCA

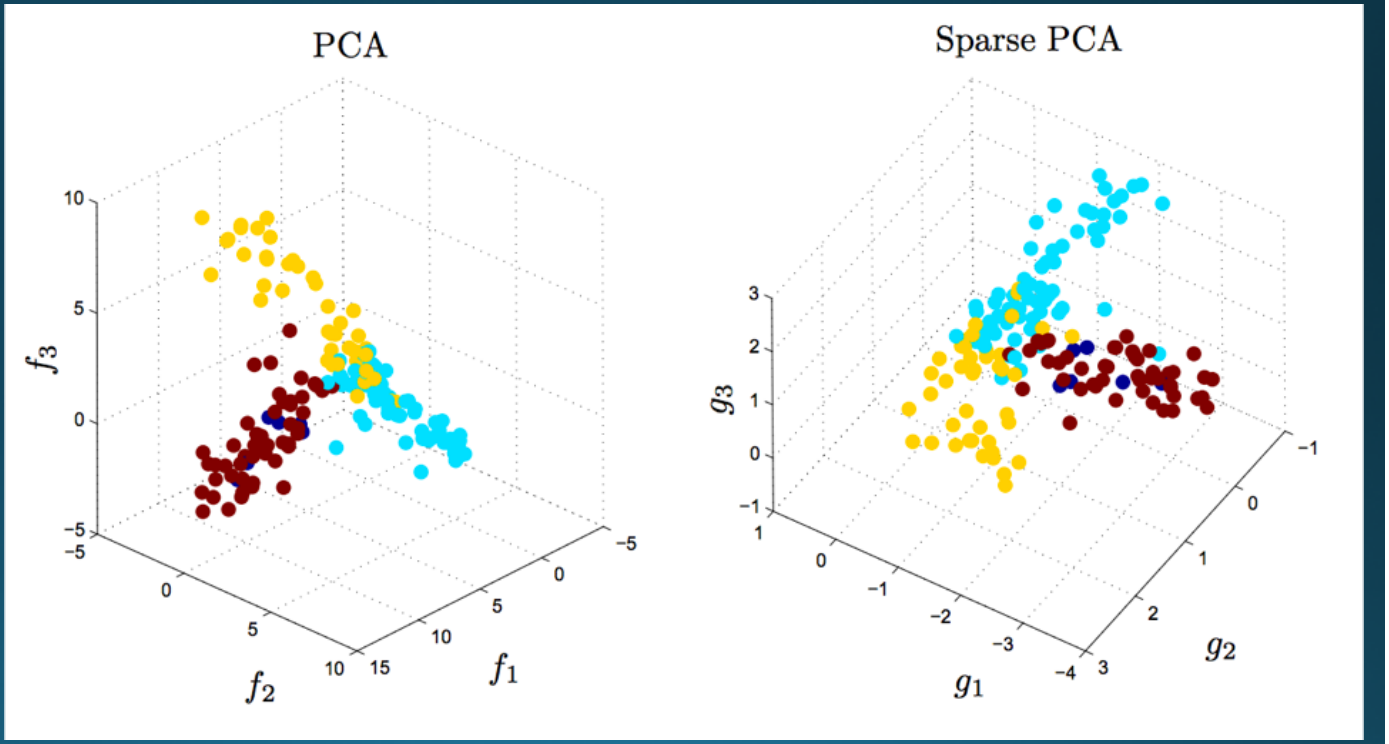
- We still want to maximize $u_i^T S u_i$, subject to $u_i^T u_i = 1$
- …and one more constraint: we want to *minimize* $||u_i||_1$

- Formalize these constraints using Lagrangian multipliers

$$\min_{W,U} ||X - WU^T||_F^2 + \gamma \sum_{n=1}^{N} ||\vec{w}_i||_1 + \gamma \sum_{i=1}^{D} ||\vec{u}_i||_1$$

Sparse PCA

- Qualitatively similar to PCA, but with lots more zeros



Sparse PCA

Advantages

- Simpler and more interpretable components
- Resulting components are very similar to "standard" PCA

Disadvantages

- Optimization procedure is non-convex (often use some version of alternating least-squares)

Summary

- Principal Components Analysis
  - Classic dimensionality reduction technique
- Kernel PCA
  - Introduces nonlinearities into component vectors
  - Permits use of arbitrary similarity functions
  - Can capture much richer and more complex interactions in data
  - Much more expensive to compute than PCA
- Sparse PCA
  - Qualitatively similar results to PCA
  - Components are sparse, improving interpretability
  - Learning procedure is non-convex, typically requiring ALS

Questions?

Resources

- http://alexhwilliams.info/itsneuronalblog/2016/03/27/pca/
- *Elements of Statistical Learning*, Chapter 14 http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
- *Pattern Recognition and Machine Learning*, Chapter 12
- *Machine Learning: A Probabilistic Perspective*, Chapter 14
- *An Introduction to Statistical Learning*, Chapter 10 http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Seventh%20Printing.pdf