Machine Learning - MT 2018
17 & 18. Dimensionality Reduction

James Worrell

University of Oxford
November 20 & 22, 2018

Supervised Learning: Summary

- Training data is of the form $\langle(\mathbf{x}_i, y_i)\rangle$ where $\mathbf{x}_i$ are features and y_i is target
- We formulate a model: generative or discriminative
- Choose a suitable training criterion (loss function, maximum likelihood)
- Use optimisation procedure to learn parameters
- Use regularization or other techniques to reduce overfitting
- Use trained classifier to predict targets/labels on unseen $\mathbf{x}_{\text{new}}$

Unsupervised Learning

Training data is of the form $\mathbf{x}_1, \ldots, \mathbf{x}_N$

Infer properties about the data

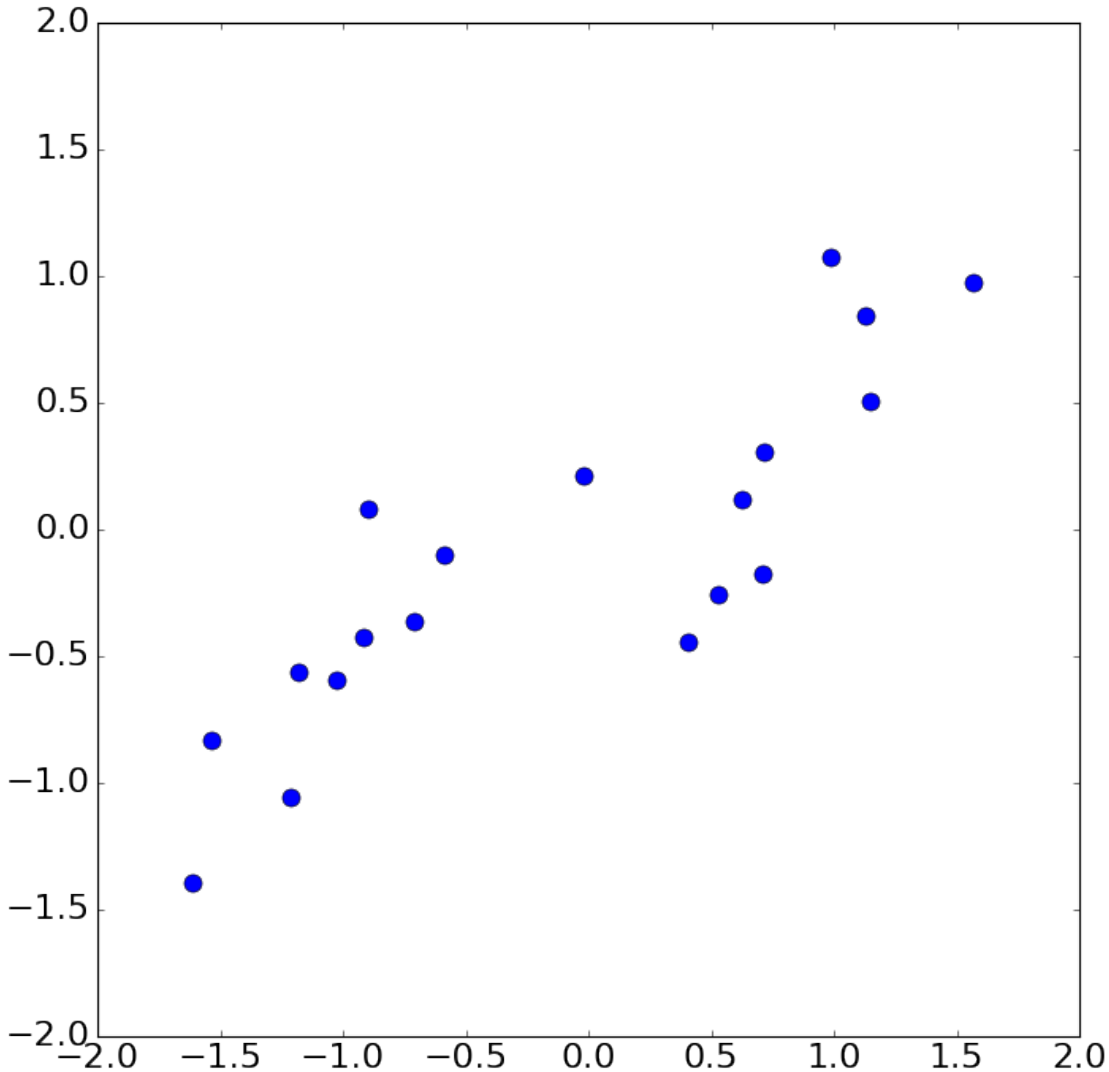
- Search: Identify patterns in data
- Density Estimation: Learn the underlying distribution generating data
- Clustering: Group similar points together
- Today: Dimensionality Reduction

Outline

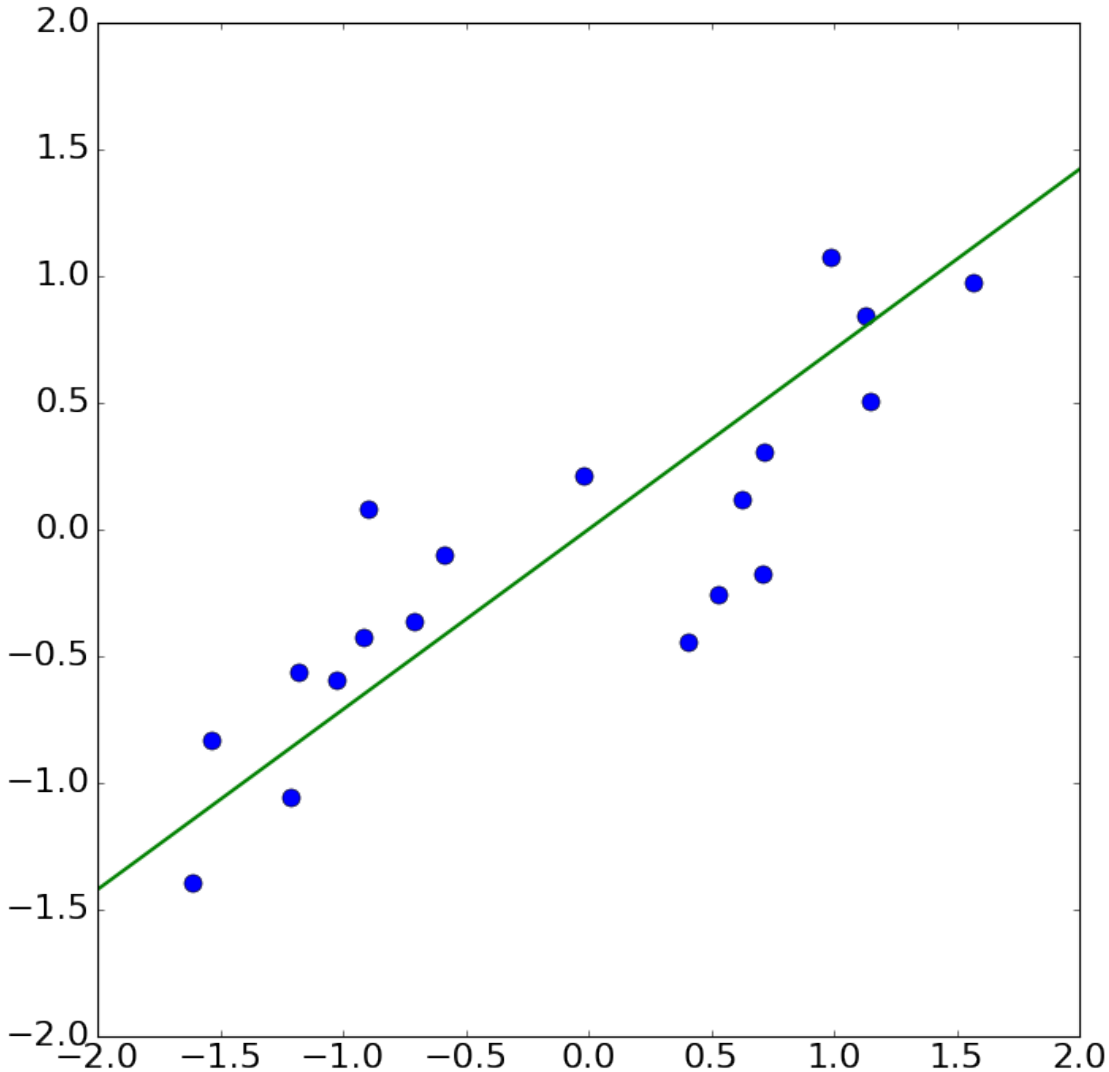
Today, we'll study a technique for dimensionality reduction

- Principal Component Analysis (PCA) identifies a small number of directions which explain most variation in the data
- PCA can be kernelised
- Dimensionality reduction is important both for visualising and as a preprocessing step before applying other (typically unsupervised) learning algorithms

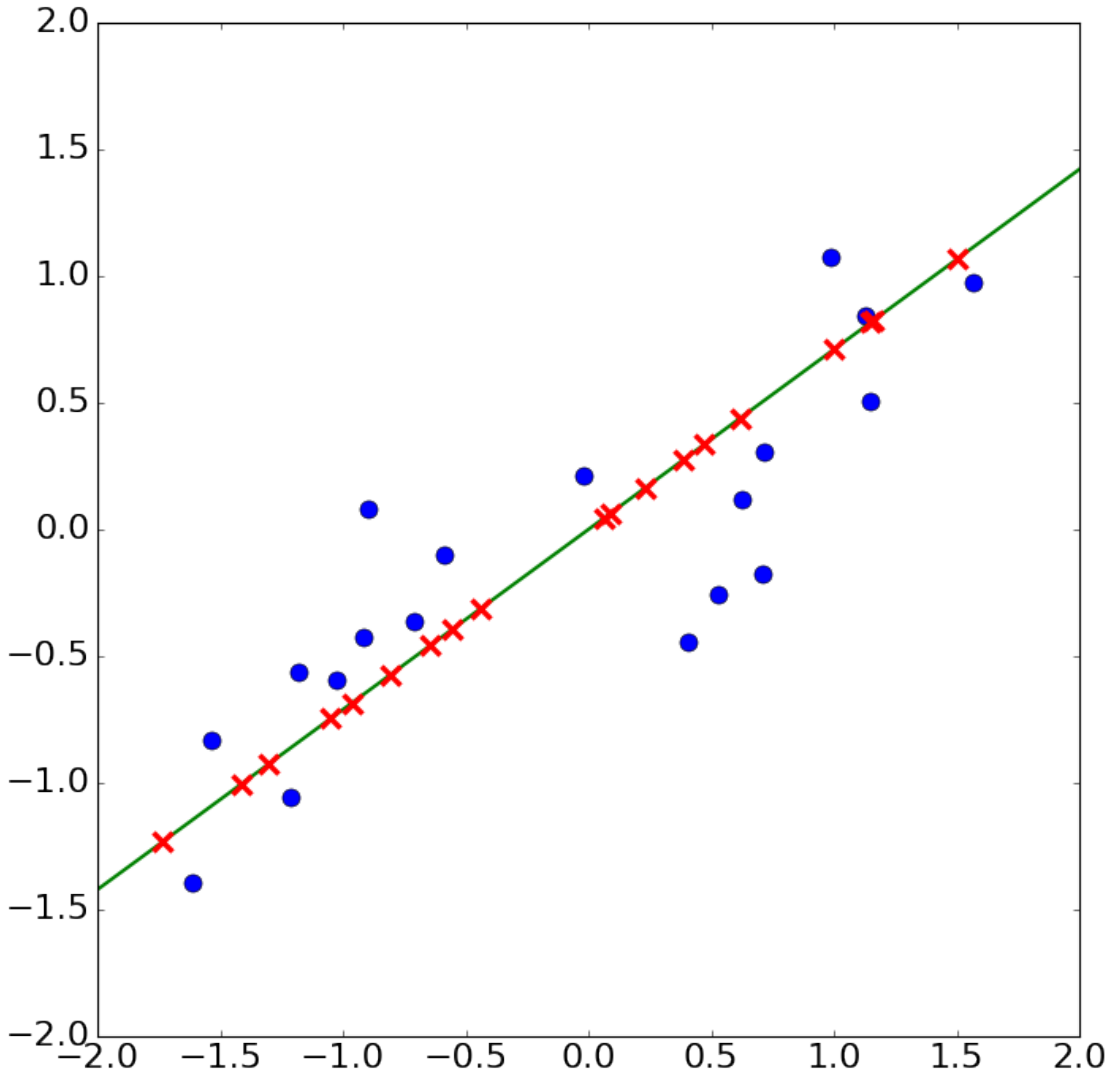
Principal Component Analysis (PCA)



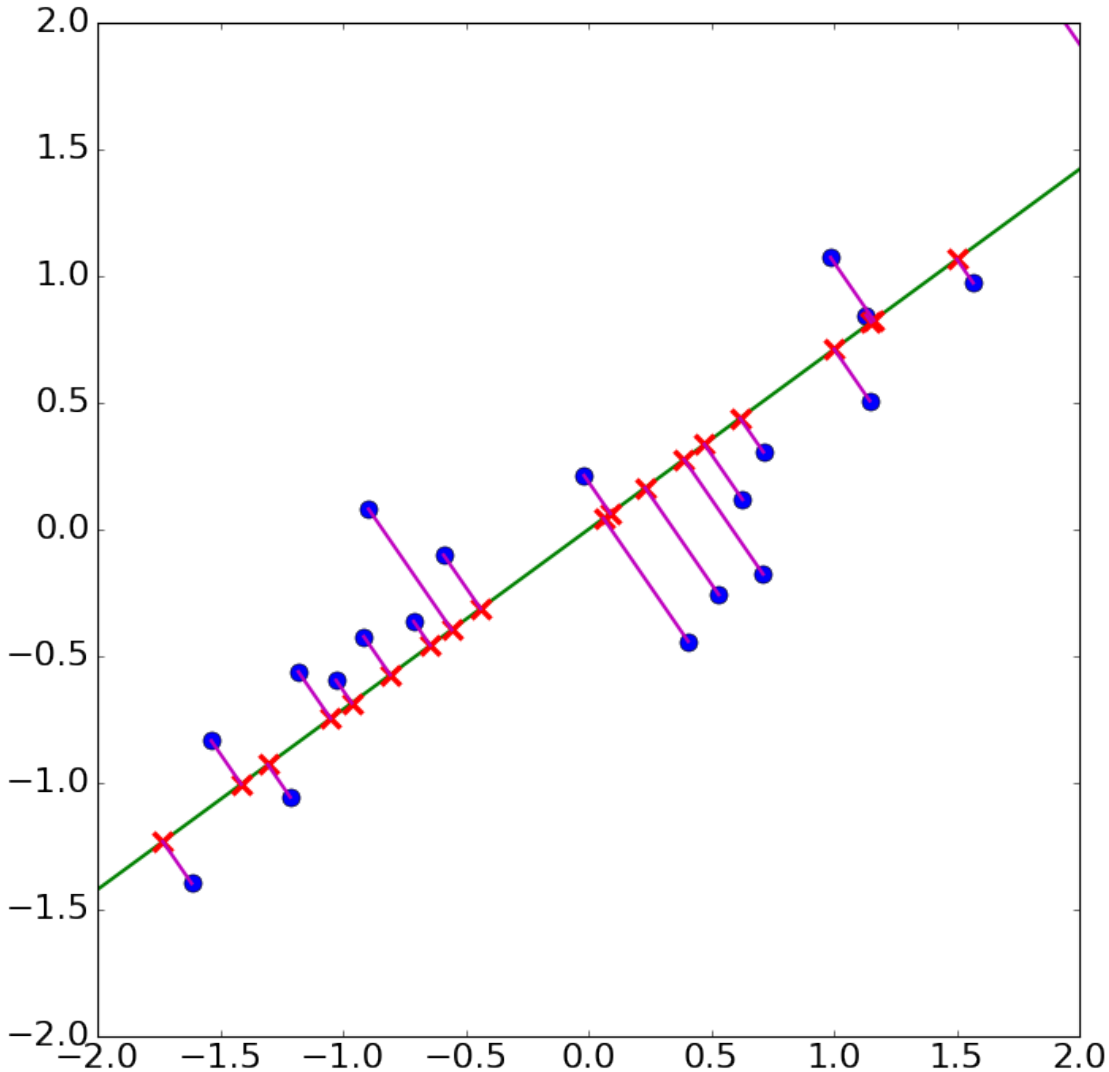
Principal Component Analysis (PCA)



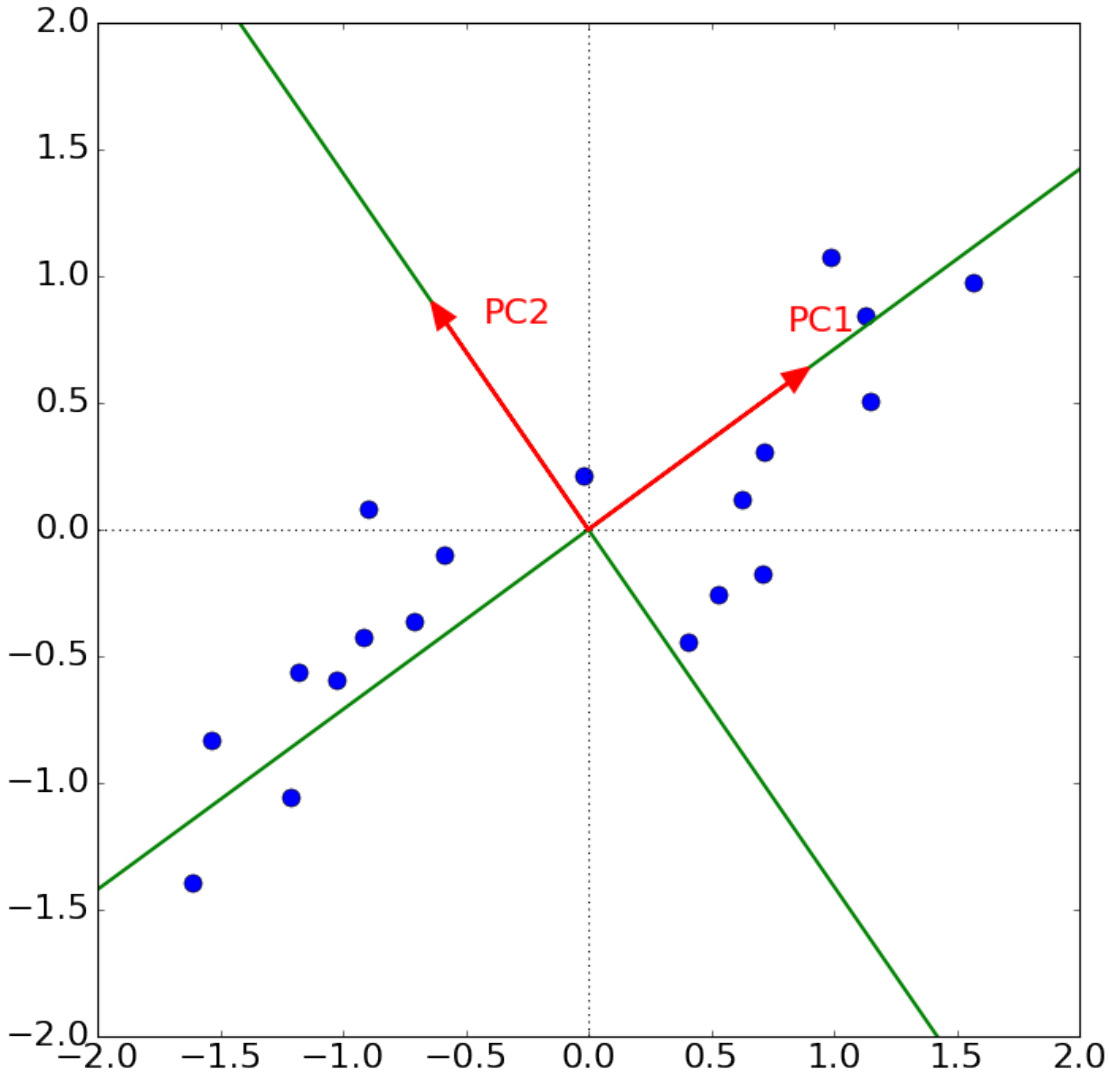
Principal Component Analysis (PCA)



Principal Component Analysis (PCA)



Principal Component Analysis (PCA)



PCA: Maximum Variance View

PCA is a *linear* dimensionality reduction technique

Find the directions of maximum variance in the data $\langle(\mathbf{x}_i)\rangle_{i=1}^N$

Assume that data is centered, *i.e.*, $\sum_i \mathbf{x}_i = \mathbf{0}$

Find a set of orthogonal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$

- The first principal component (PC) $\mathbf{v}_1$ is the direction of largest variance
- The second PC $\mathbf{v}_2$ is the direction of largest variance orthogonal to $\mathbf{v}_1$
- The i^{th} PC $\mathbf{v}_i$ is the direction of largest variance orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}$

$\mathbf{V}_{D\times k}$ gives projection

$$\mathbf{z}_i = \mathbf{V}^\mathsf{T}\mathbf{x}_i \qquad \text{for datapoint } \mathbf{x}_i$$

$$\mathbf{Z} = \mathbf{X}\mathbf{V} \qquad \text{for entire dataset}$$

PCA: Maximum Variance View

We are given i.i.d. data $\langle(\mathbf{x}_i)\rangle_{i=1}^N$; data matrix $\mathbf{X}$

Want to find $\mathbf{v}_1 \in \mathbb{R}^D$, $\|\mathbf{v}_1\| = 1$, that maximizes $\|\mathbf{X}\mathbf{v}_1\|^2$

Let $\mathbf{z} = \mathbf{X}\mathbf{v}_1$, so $z_i = \mathbf{x}_i \cdot \mathbf{v}_1$.

We wish to find $\mathbf{v}_1$ so that $\sum_{i=1}^N z_i^2$ is maximised.

$$\sum_{i=1}^{N} z_i^2 = \mathbf{z}^\mathsf{T}\mathbf{z}$$

$$= \mathbf{v}_1^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{v}_1$$

The maximum value attained by $\mathbf{v}_1^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{v}_1$ for $\|\mathbf{v}_1\| = 1$ is the largest eigenvalue of $\mathbf{X}^\mathsf{T}\mathbf{X}$.

The argmax is the corresponding eigenvector $\mathbf{v}_1$.

Find $\mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_k$ that are all successively orthogonal to previous directions and maximise (as yet unexplained variance)

PCA: Best Reconstruction

We have i.i.d. data $\langle(\mathbf{x}_i)\rangle_{i=1}^N$; data matrix $\mathbf{X}$

Find a k-dimensional linear projection that best represents the data

Suppose $\mathbf{V}_k \in \mathbb{R}^{D\times k}$ is such that columns of $\mathbf{V}_k$ are orthogonal

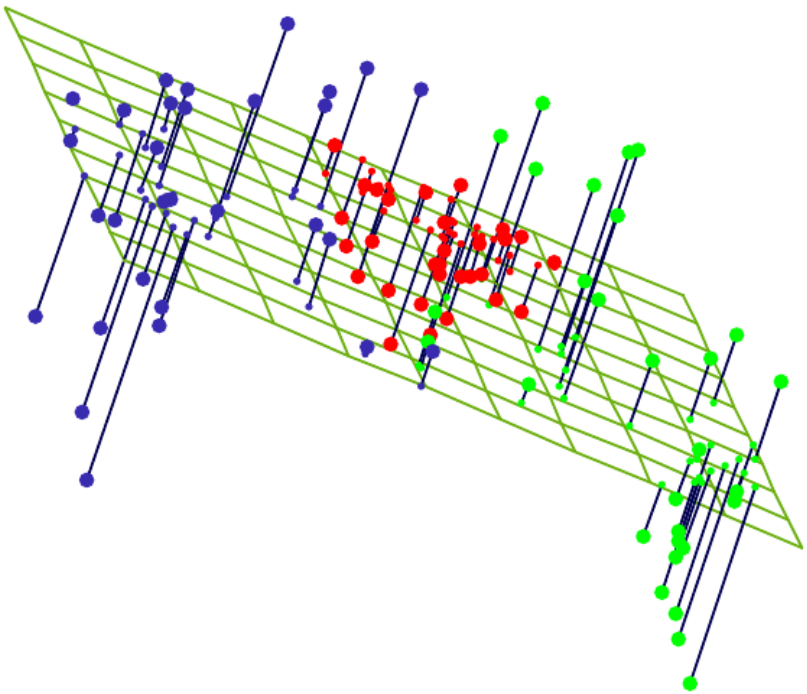
Project data $\mathbf{X}$ on to subspace defined by $\mathbf{V}$

$$\mathbf{Z} = \mathbf{X}\mathbf{V}_k$$

Minimize reconstruction error

$$\sum_{i=1}^{N} \left\|\mathbf{x}_i - \mathbf{V}_k\mathbf{V}_k^{\mathsf{T}}\mathbf{x}_i\right\|^2$$

Principal Component Analysis (PCA)



Equivalence between the Two Objectives: One PC Case

Let $\mathbf{v}_1$ be the direction of projection

The point $\mathbf{x}$ is mapped to $\tilde{\mathbf{x}} = (\mathbf{v}_1 \cdot \mathbf{x})\mathbf{v}_1$, where $\|\mathbf{v}_1\| = 1$

Maximum Variance

Find $\mathbf{v}_1$ that maximises $\sum_{i=1}^{N} (\mathbf{v}_1 \cdot \mathbf{x}_i)^2$

Best Reconstruction

Find $\mathbf{v}_1$ that minimises:

$$\begin{aligned}
\sum_{i=1}^{N} \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2 &= \sum_{i=1}^{N} \left(\|\mathbf{x}_i\|^2 - 2(\mathbf{x}_i \cdot \tilde{\mathbf{x}}_i) + \|\tilde{\mathbf{x}}_i\|^2 \right) \\
&= \sum_{i=1}^{N} \left(\|\mathbf{x}_i\|^2 - 2(\mathbf{v}_1 \cdot \mathbf{x}_i)^2 + (\mathbf{v}_1 \cdot \mathbf{x}_i)^2 \|\mathbf{v}_1\|^2 \right) \\
&= \sum_{i=1}^{N} \|\mathbf{x}_i\|^2 - \sum_{i=1}^{N} (\mathbf{v}_1 \cdot \mathbf{x}_i)^2
\end{aligned}$$

So the same $\mathbf{v}_1$ satisfies the two objectives

Finding Principal Components: SVD

Let $\mathbf{X}$ be the $N \times D$ data matrix

Pair of singular vectors $\mathbf{u} \in \mathbb{R}^N$, $\mathbf{v} \in \mathbb{R}^D$ and singular value $\sigma \in \mathbb{R}^+$ if

$$\sigma \mathbf{u} = \mathbf{X}\mathbf{v} \quad \text{and} \quad \sigma \mathbf{v} = \mathbf{X}^\mathsf{T}\mathbf{u}$$

$\mathbf{v}$ is an eigenvector of $\mathbf{X}^\mathsf{T}\mathbf{X}$ with eigenvalue σ^2

$\mathbf{u}$ is an eigenvector of $\mathbf{X}\mathbf{X}^\mathsf{T}$ with eigenvalue σ^2

Finding Principal Components: SVD

$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ (say $N > D$)

Thin SVD: $\mathbf{U}$ is $N \times D$, $\mathbf{\Sigma}$ is $D \times D$, $\mathbf{V}$ is $D \times D$, $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{I}_D$

$\mathbf{\Sigma}$ is diagonal with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_D \geq 0$

The first k principal components are first k columns of $\mathbf{V}$

Full SVD: $\mathbf{U}$ is $N \times N$, $\mathbf{\Sigma} = N \times D$, $\mathbf{V}$ is $D \times D$. $\mathbf{V}$ and $\mathbf{U}$ are orthonormal matrices

Algorithm for finding PCs (when N > D)

Constructing the matrix $\mathbf{X}^\mathsf{T}\mathbf{X}$ takes time $O(D^2N)$

Eigenvectors of $\mathbf{X}^\mathsf{T}\mathbf{X}$ can be computed in time $O(D^3)$

Iterative methods to get top k singular (right) vectors directly:

- Initiate $\mathbf{v}^0$ to be random unit norm vector
- Iterative Update:
  - $\mathbf{v}^{t+1} = \mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{v}^t$
  - $\mathbf{v}^{t+1} = \mathbf{v}^{t+1}/\|\mathbf{v}^{t+1}\|$

  until (approximate) convergence
- Update step only takes $O(ND)$ time (compute $\mathbf{X}\mathbf{v}^t$ first, then $\mathbf{X}^\mathsf{T}(\mathbf{X}\mathbf{v}^t)$)
- This gives the singular vector corresponding to the largest singular value
- Subsequent singular vectors obtained by choosing $\mathbf{v}^0$ orthogonal to previously identified singular vectors (this needs to be done at each iteration to avoid numerical errors creeping in)

Algorithm for finding PCs (when D $\gg$ N)

Constructing the matrix $\mathbf{X}\mathbf{X}^\mathsf{T}$ takes time $O(N^2D)$

Eigenvectors of $\mathbf{X}\mathbf{X}^\mathsf{T}$ can be computed in time $O(N^3)$

The eigenvectors give the 'left' singular vectors, $\mathbf{u}_i$ of $\mathbf{X}$

To obtain $\mathbf{v}_i$, we use the fact that $\mathbf{v}_i = \sigma^{-1}\mathbf{X}^\mathsf{T}\mathbf{u}_i$

Iterative method can be used directly as in the case when $N > D$

PCA: Reconstruction Error

We have thin SVD: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$

Let $\mathbf{V}_k$ be the matrix containing first k columns of $\mathbf{V}$

Projection on to k PCs: $\mathbf{Z} = \mathbf{X}\mathbf{V}_k = \mathbf{U}_k\mathbf{\Sigma}_k$, where $\mathbf{U}_k$ is the matrix of the first k columns of $\mathbf{U}$ and $\mathbf{\Sigma}_k$ is the $k \times k$ diagonal submatrix for $\mathbf{\Sigma}$ of the top k singular values

Reconstruction: $\tilde{\mathbf{X}} = \mathbf{Z}\mathbf{V}_k^\mathsf{T} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\mathsf{T}$

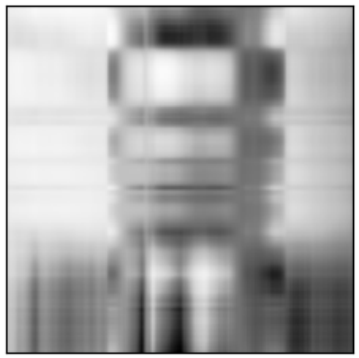
Reconstruction error $= \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{V}_k\mathbf{V}_k^\mathsf{T}\mathbf{x}_i\|^2 = \sum_{j=k+1}^{D} \sigma_j^2$

This follows from the following calculations:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T} = \sum_{j=1}^{D} \sigma_j \mathbf{u}_j \mathbf{v}_j^\mathsf{T} \qquad \widetilde{\mathbf{X}} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\mathsf{T} = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^\mathsf{T}$$

$$\|\mathbf{X} - \tilde{\mathbf{X}}\|_F = \sum_{j=k+1}^{D} \sigma_j^2$$

Reconstruction of an Image using PCA



K = 2

K = 5

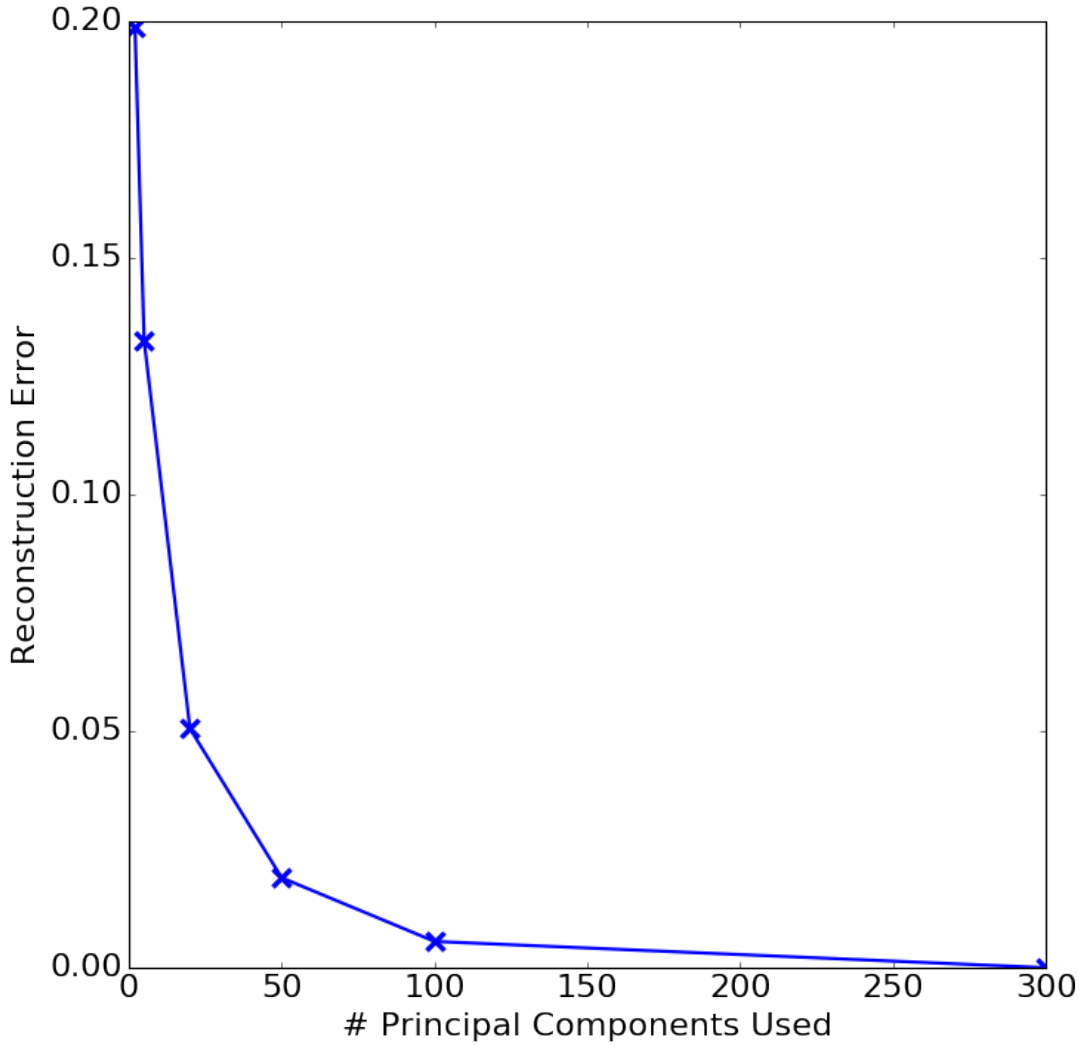
K = 20

K = 50

K = 100

K = 300

How many principal components to pick?



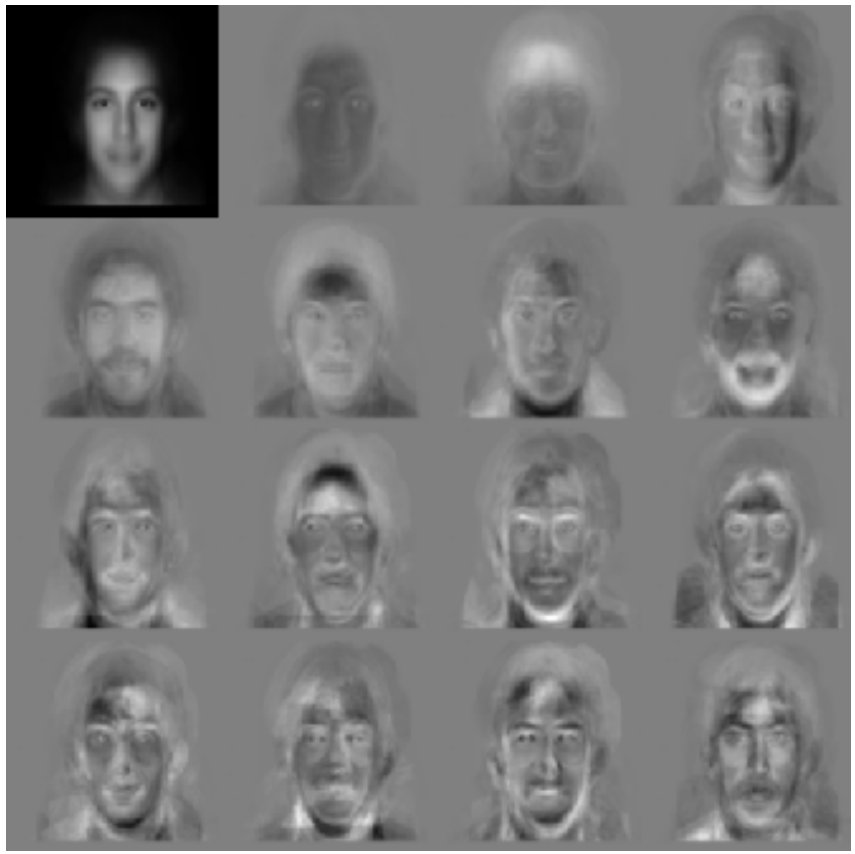
Look for an ‘elbow’ in the curve of reconstruction error vs # PCs

Application: Eigenfaces

A popular application of PCA for face detection and recognition is known as Eigenfaces

- Face detection: Identify faces in a given image
- Face Recognition: Classification (or search) problem to identify a certain person

Application: Eigenfaces



PCA on a dataset of face images. Each principal component can be thought of as being an ‘element’ of a face.

Source: http://vismod.media.mit.edu/vismod/demos/facerec/basic.html

Application: Eigenfaces

Detection: Each patch of the image can be checked to identify whether there is a face in it

Recognition: Map all faces in terms of their principal components. Then use some distance measure on the projections to find faces that are most like the input image.

Why use PCA for face detection?

- Even though images can be large, we can use the $D \gg N$ approach to be efficient
- The final model (the PCs) can be quite compact, can fit on cameras, phones
- Works very well given the simplicity of the model

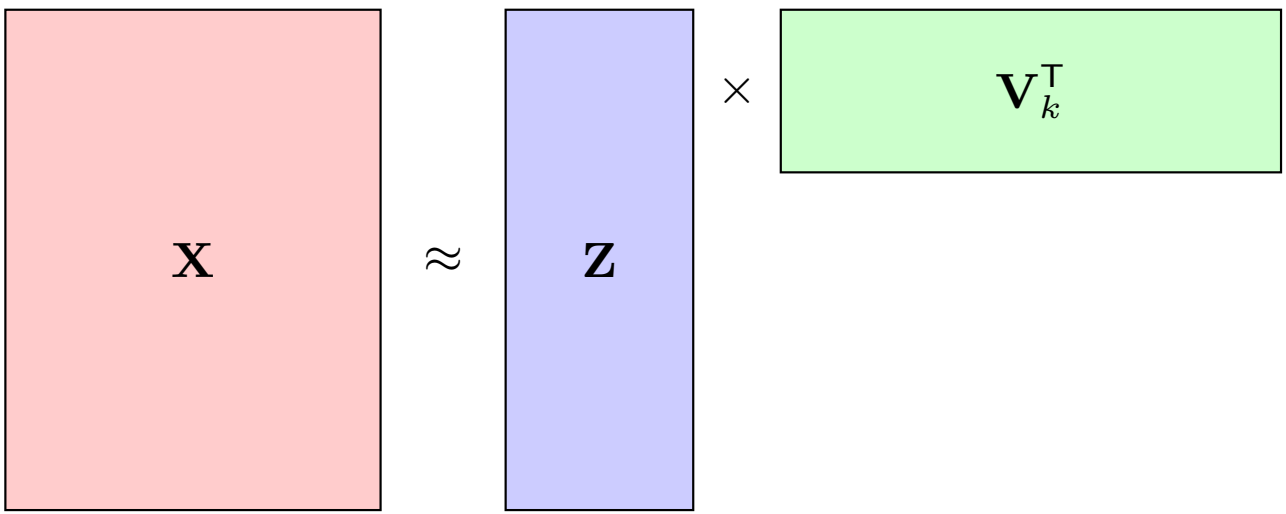
Application: Latent Semantic Analysis

$\mathbf{X}$ is an $N \times D$ matrix, D is the size of dictionary

$\mathbf{x}_i$ is a vector of word counts (bag of words)

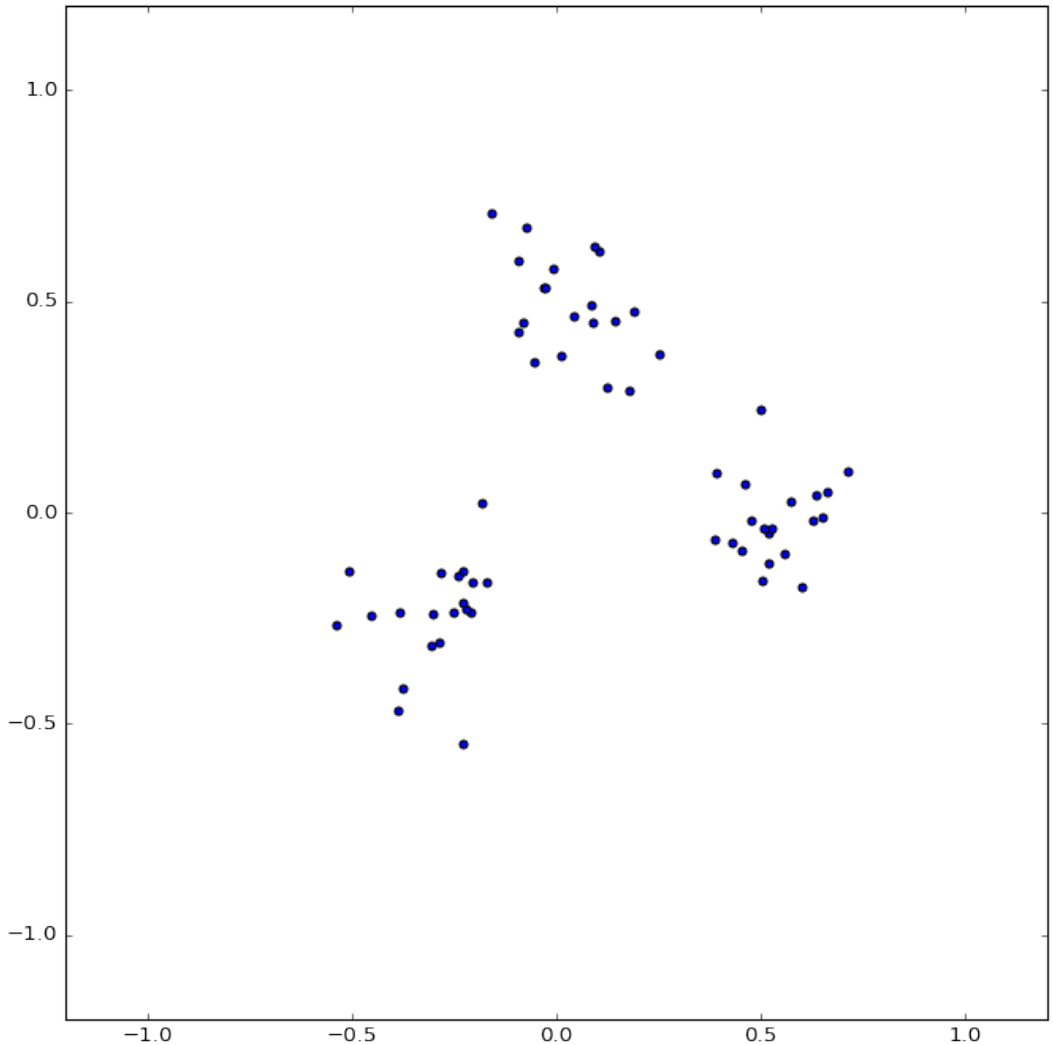
Reconstruction using k eigenvectors $\mathbf{X} \approx \mathbf{Z}\mathbf{V}_k^\mathsf{T}$, where $\mathbf{Z} = \mathbf{X}\mathbf{V}_k$

$\langle \mathbf{z}_i, \mathbf{z}_j \rangle$ is probably a better notion of similarity than $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$

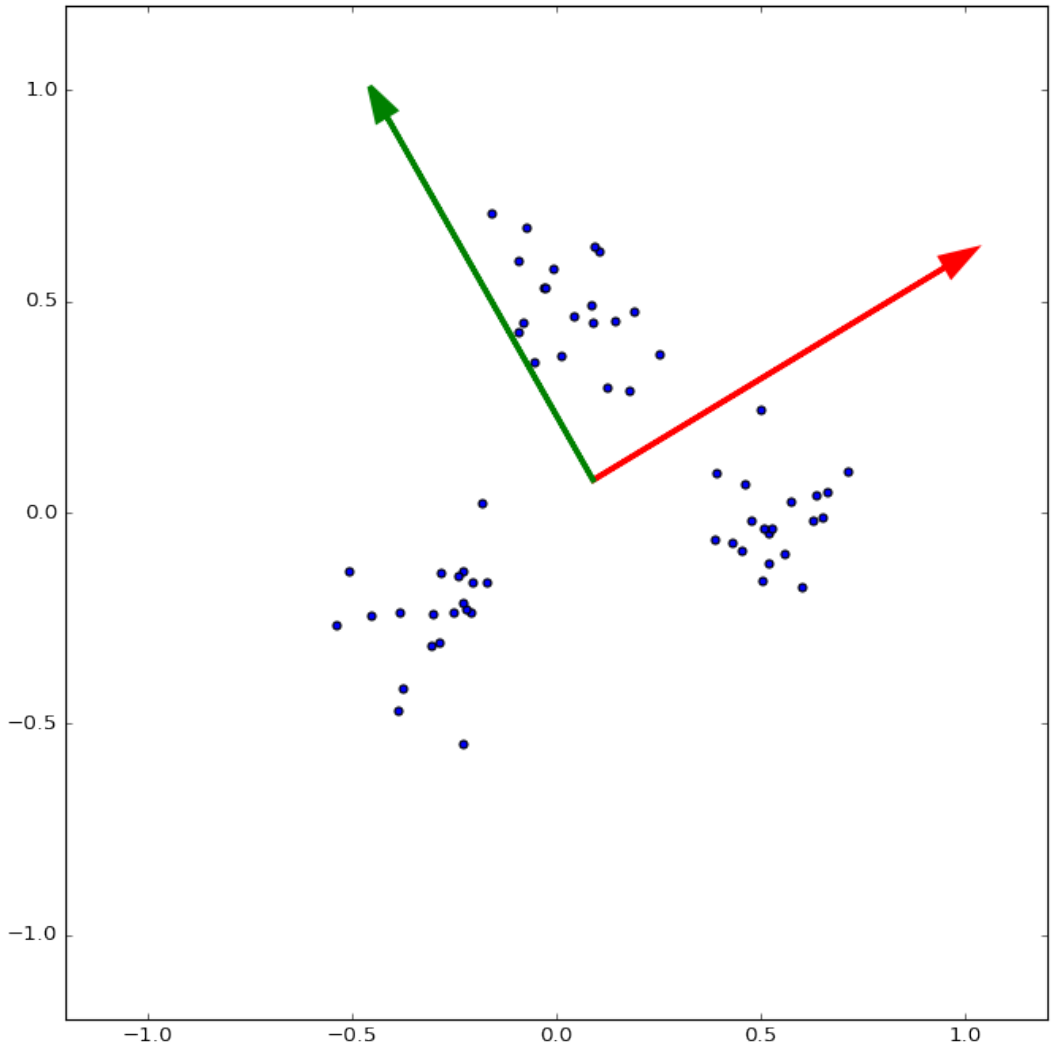


Non-negative matrix factorisation has more natural interpretation, but is harder to compute

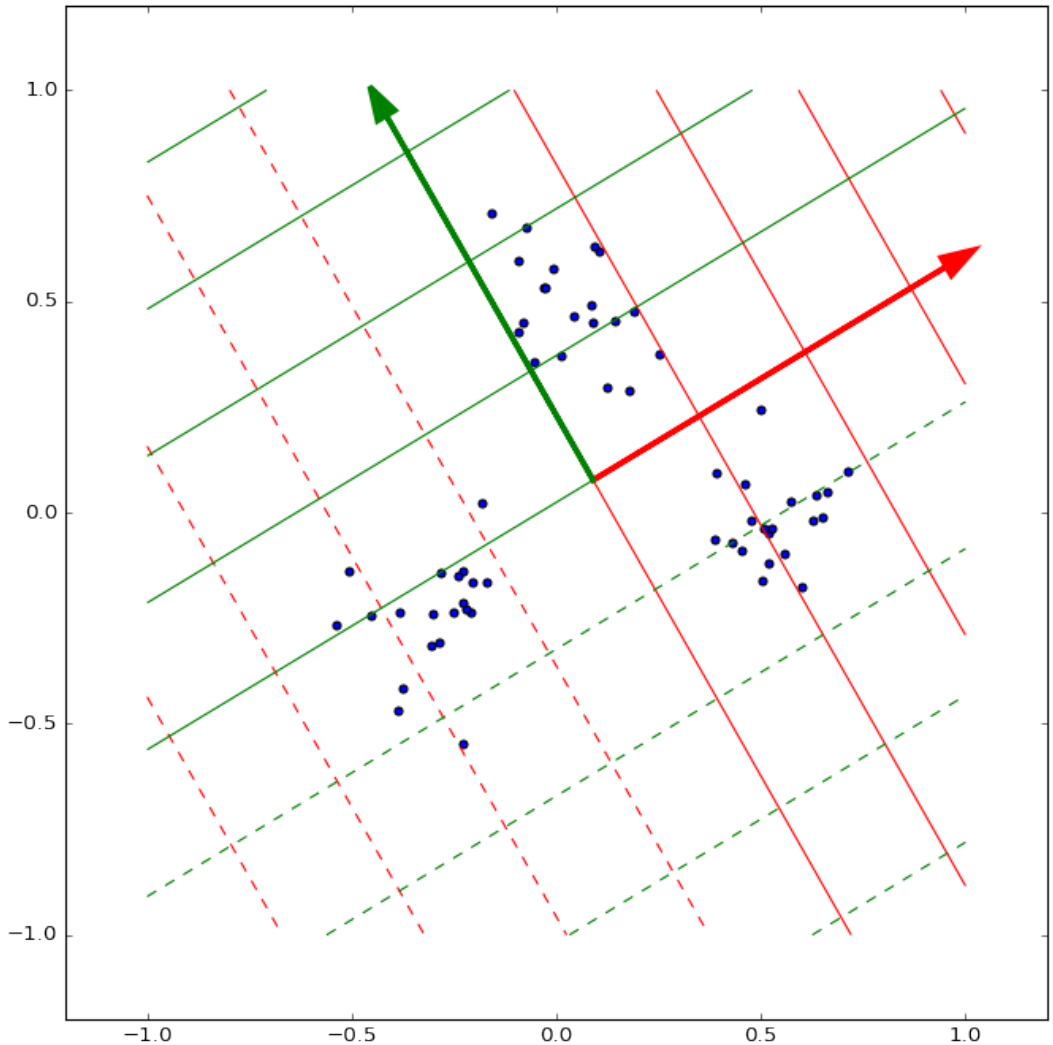
PCA: Beyond Linearity



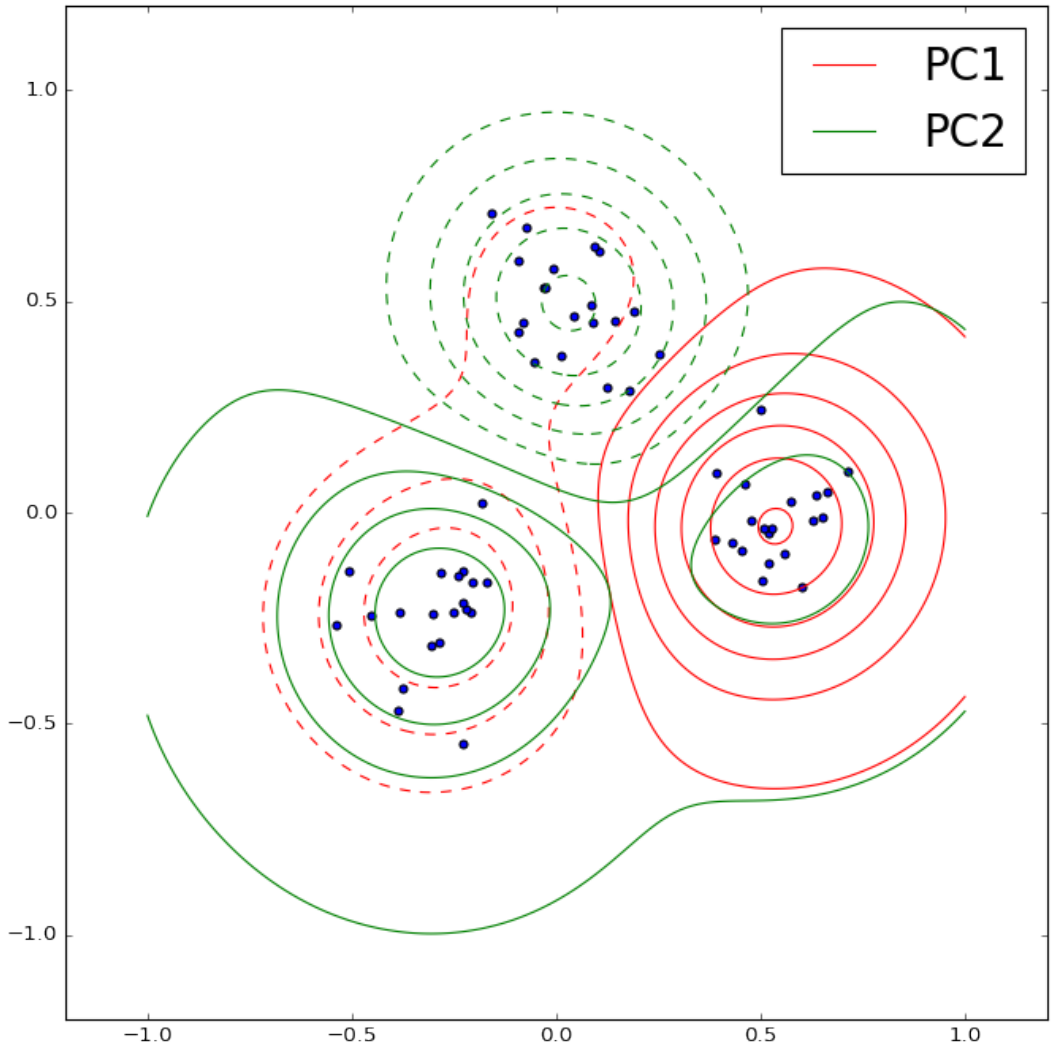
PCA: Beyond Linearity



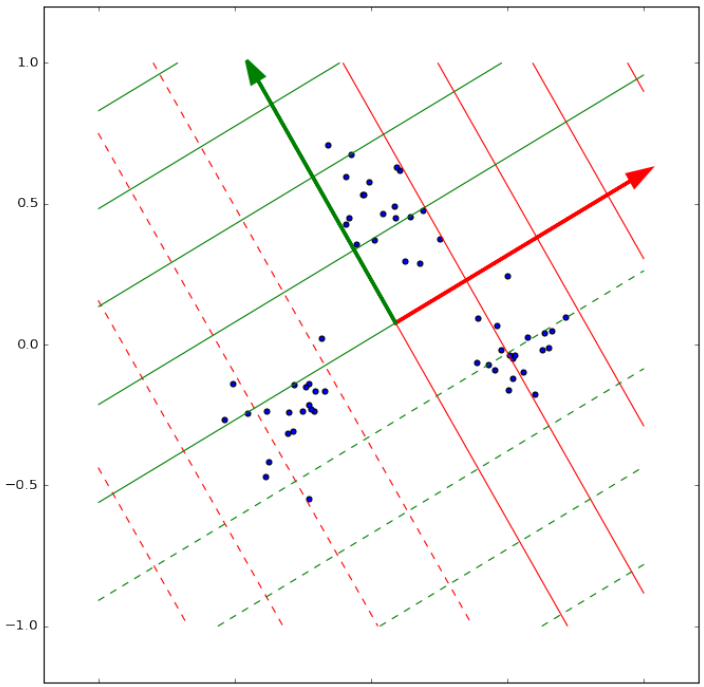
PCA: Beyond Linearity

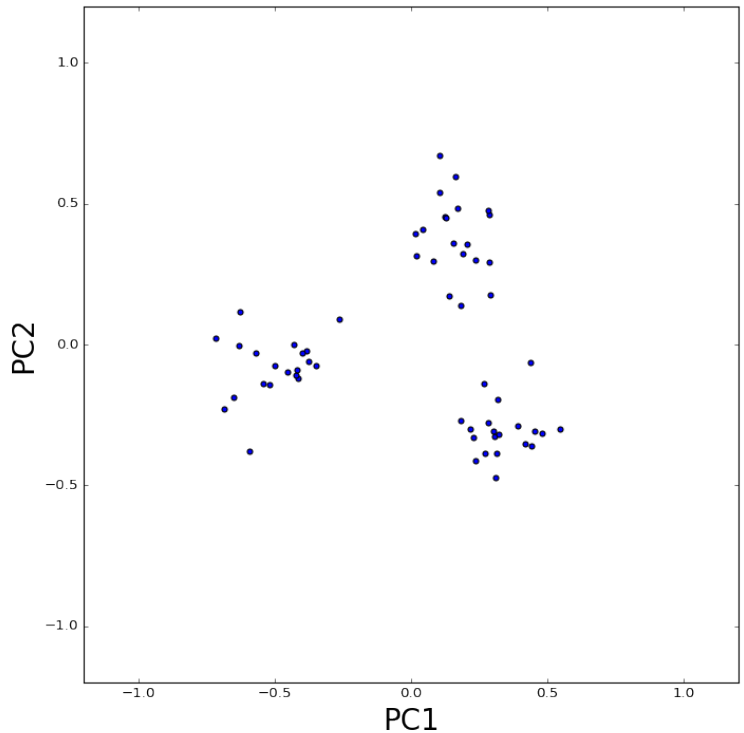


PCA: Beyond Linearity

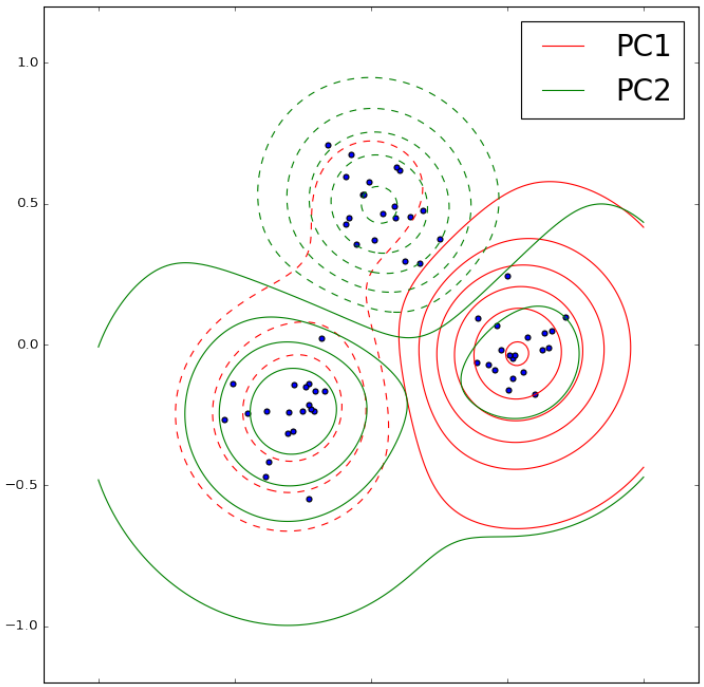


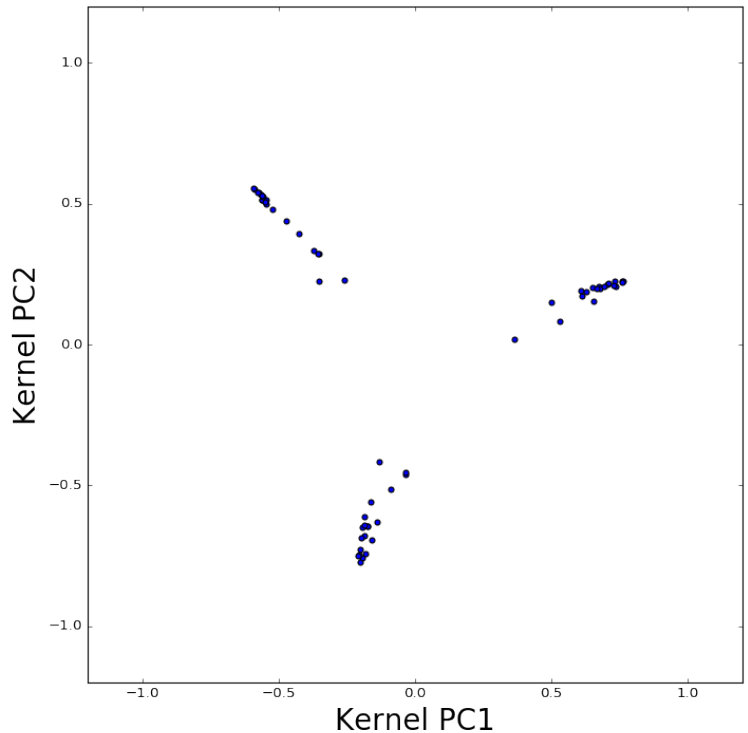
Projection: Linear PCA





Projection: Kernel PCA





Kernel PCA

Suppose our original data is, for example, $\mathbf{x} \in \mathbb{R}^2$

We could perform degree 2 polynomial basis expansion as:

$$\phi(\mathbf{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right]^\mathsf{T}$$

Recall that we can compute the inner products $\phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$ efficiently using the kernel trick

$$\begin{aligned}\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') &= 1 + 2x_1x_1' + 2x_2x_2' + x_1^2(x_1')^2 + x_2^2(x_2')^2 + 2x_1x_2x_1'x_2' \\ &= (1 + x_1x_2 + x_1'x_2')^2 = (1 + \mathbf{x} \cdot \mathbf{x}')^2 =: \kappa(\mathbf{x}, \mathbf{x}')\end{aligned}$$

Kernel PCA

Suppose we use the feature map: $\phi : \mathbb{R}^D \to \mathbb{R}^M$

Let $\phi(\mathbf{X})$ be the $N \times M$ matrix

We want find the singular vectors of $\phi(\mathbf{X})$ (eigenvectors of $\phi(\mathbf{X})^\mathsf{T}\phi(\mathbf{X})$)

However, in general $M \gg N$ (in fact M could be infinite for some kernels)

Instead we'll find the eigenvectors of $\phi(\mathbf{X})\phi(\mathbf{X})^\mathsf{T}$, the kernel matrix

Kernel PCA

Recall that the kernel matrix is:

$$\mathbf{K} = \phi(\mathbf{X})\phi(\mathbf{X})^{\mathsf{T}} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \kappa(\mathbf{x}_N, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

Let $\mathbf{u} \in \mathbb{R}^N$ be an eigenvector of $\mathbf{K}$, (left singular vector of $\phi(\mathbf{X})$)

The corresponding principal component $\mathbf{v} \in \mathbb{R}^M$ is $\sigma^{-1}\phi(\mathbf{X})^{\mathsf{T}}\mathbf{u}$

We won't express $\mathbf{v}$ explicitly, instead we can compute projections of a new datapoint $\mathbf{x}_{\text{new}}$ on to the principal component $\mathbf{v}$ using the kernel function:

$$\phi(\mathbf{x}_{\text{new}})^{\mathsf{T}}\mathbf{v} = \sigma^{-1}\phi(\mathbf{x}_{\text{new}})^{\mathsf{T}}\phi(\mathbf{X})^{\mathsf{T}}\mathbf{u} \quad = \sigma^{-1}[\kappa(\mathbf{x}_{\text{new}}, \mathbf{x}_1), \kappa(\mathbf{x}_{\text{new}}, \mathbf{x}_2), \cdots, \kappa(\mathbf{x}_{\text{new}}, \mathbf{x}_N)]\mathbf{u}$$

So in order to compute projections onto principal components we do not need to store the principal components explicitly!

Kernel PCA

For PCA, we assumed that the datamatrix $\mathbf{X}$ is centered, *i.e.*, $\sum_i \mathbf{x}_i = \mathbf{0}$

However, this is not the case for the matrix $\phi(\mathbf{X})$

Instead we can consider:

$$\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \frac{1}{N}\sum_{k=1}^{N}\phi(\mathbf{x}_k)$$

The corresponding matrix $\tilde{\mathbf{K}}$ is given by the entries

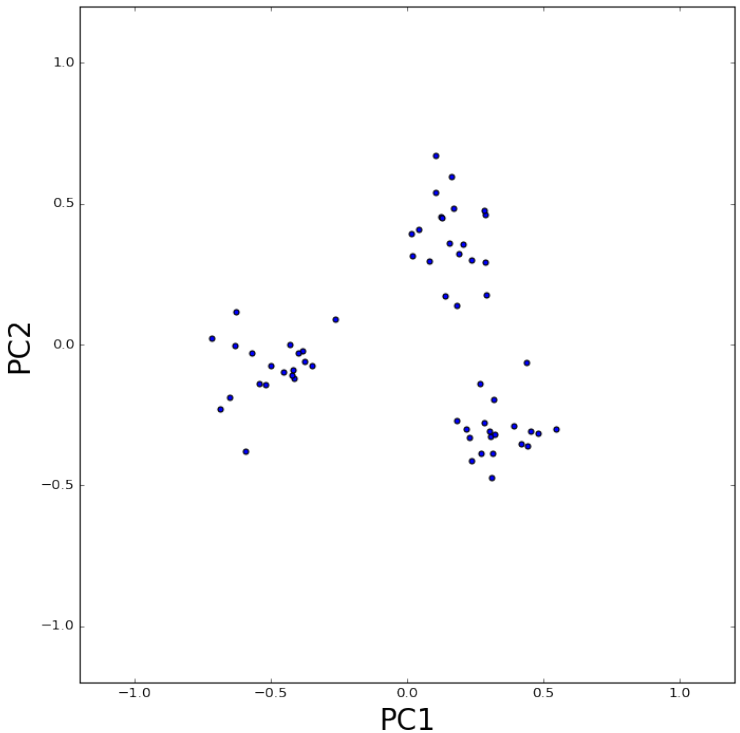
$$\tilde{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{N}\sum_{l=1}^{N}\kappa(\mathbf{x}_i, \mathbf{x}_l) - \frac{1}{N}\sum_{l=1}^{N}\kappa(\mathbf{x}_j, \mathbf{x}_l) + \frac{1}{N^2}\sum_{k=1}^{N}\sum_{l=1}^{N}\kappa(\mathbf{x}_l, \mathbf{x}_k)$$

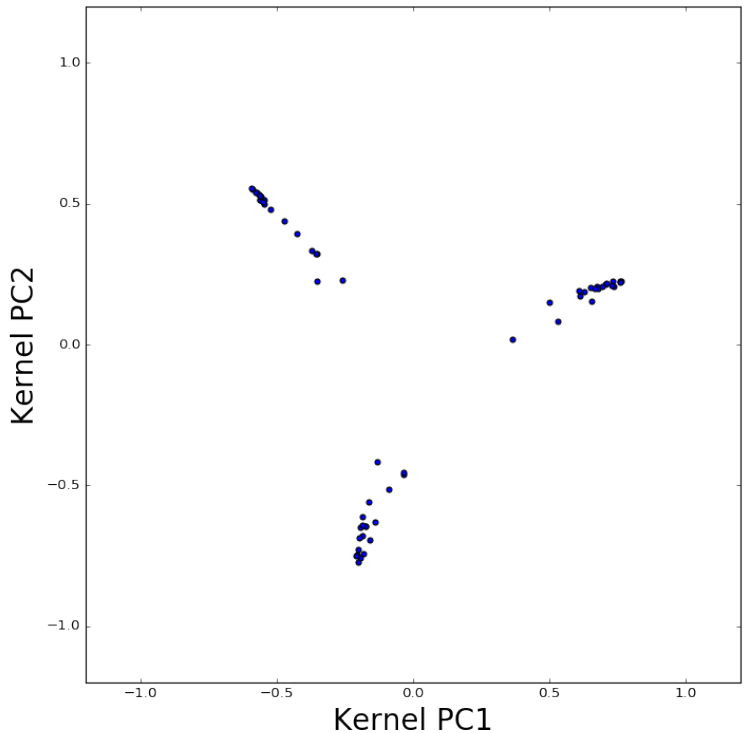
Succintly, if $\mathbf{O}$ is the matrix of all with every entry $1/N$, *i.e.*, $\mathbf{O} = \mathbf{1}\mathbf{1}^{\mathsf{T}}/N$

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{O}\mathbf{K} - \mathbf{K}\mathbf{O} + \mathbf{O}\mathbf{K}\mathbf{O}$$

To perform kernel PCA, we need to find the eigenvectors of $\tilde{\mathbf{K}}$

Projection: PCA vs Kernel PCA





Kernel PCA Applications

- Kernel PCA is not necessarily very useful for visualisation
- Also, kernel PCA does not directly give a useful way to construct a low-dimensional reconstruction of the original data
- Most powerful uses of kernel PCA are in other machine learning applications
- After kernel PCA preprocessing, we may get higher accuracy for classification, clustering, *etc.*

PCA Summary

Algorithm: We've expressed PCA as SVD of data matrix $\mathbf{X}$

Equivalently, we can use eigendecomposition of the matrix $\mathbf{X}^\top\mathbf{X}$

Running Time: $O(NDk)$ to compute k principal components (avoid computing the matrix $\mathbf{X}^\top\mathbf{X}$)

PCs are uncorrelated, but there may be non-linear (higher-order) effects

PCA depends on scale or units of measurement; it may be a good idea to standardize data

PCA is sensitive to outliers

PCA can be kernelised: Useful as preprocessing for further ML applications, rather than visualisation