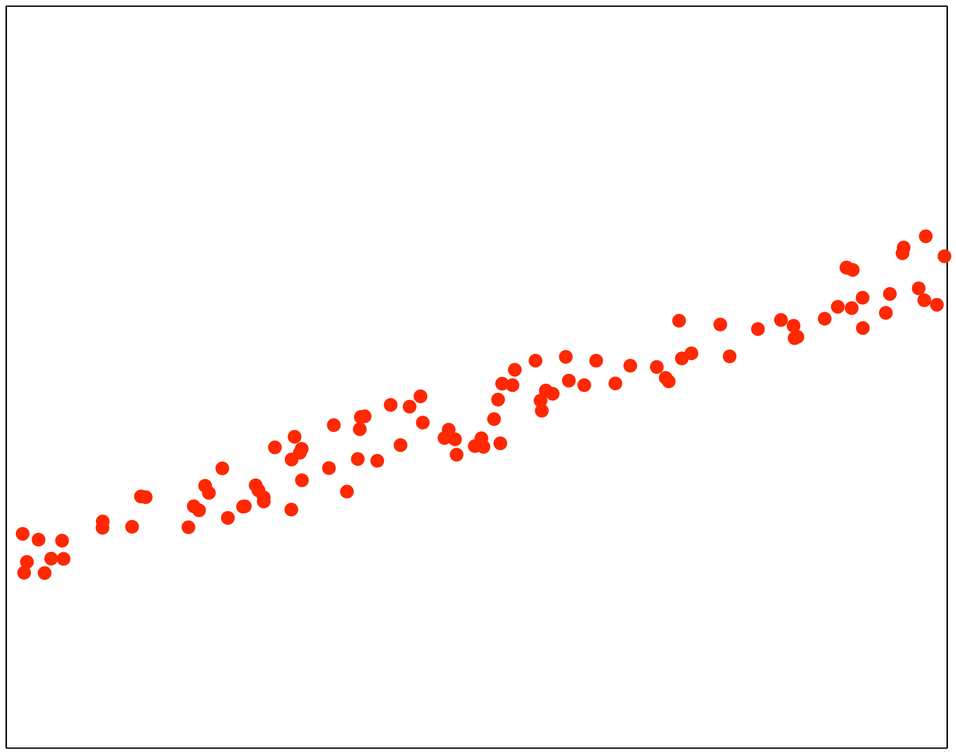
Dimensionality reduction. PCA. Kernel PCA.

- Dimensionality reduction
- Principal Component Analysis (PCA)
- Kernelizing PCA
- If we have time: Autoencoders

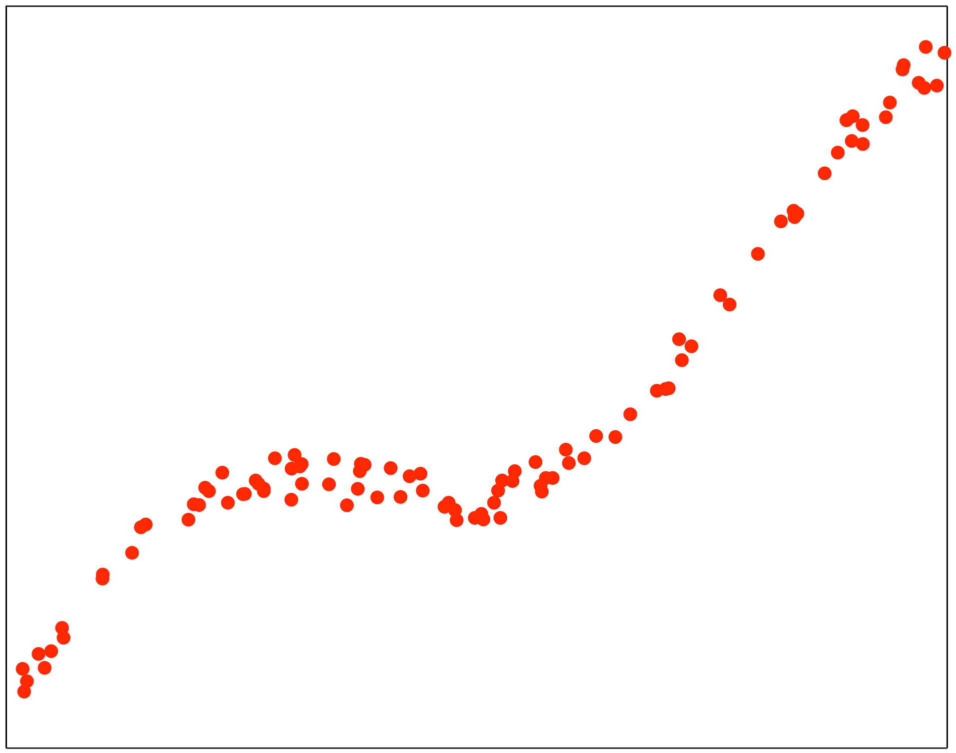
What is dimensionality reduction?

- Dimensionality reduction (or embedding) techniques:
  - Assign instances to real-valued vectors, in a space that is much smaller-dimensional (even 2D or 3D for visualization).
  - Approximately preserve similarity/distance relationships between instances.
- Some techniques:
  - Linear: Principal components analysis
  - Non-linear
    - Kernel PCA
    - Independent components analysis
    - Self-organizing maps
    - Multi-dimensional scaling
    - Autoencoders

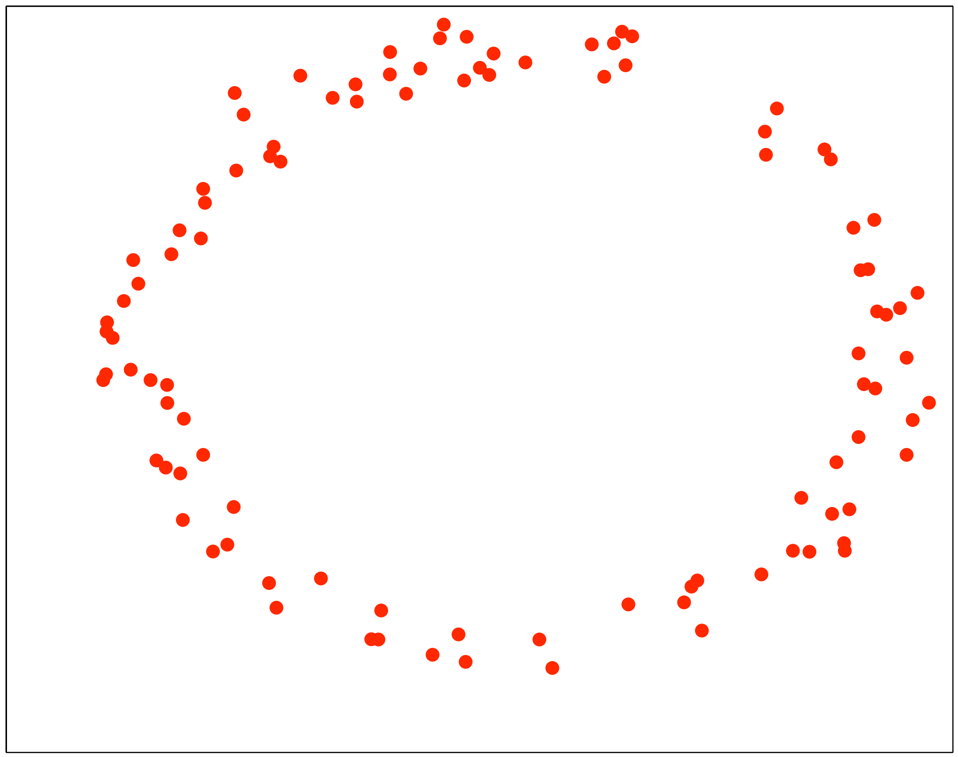
What is the true dimensionality of this data?



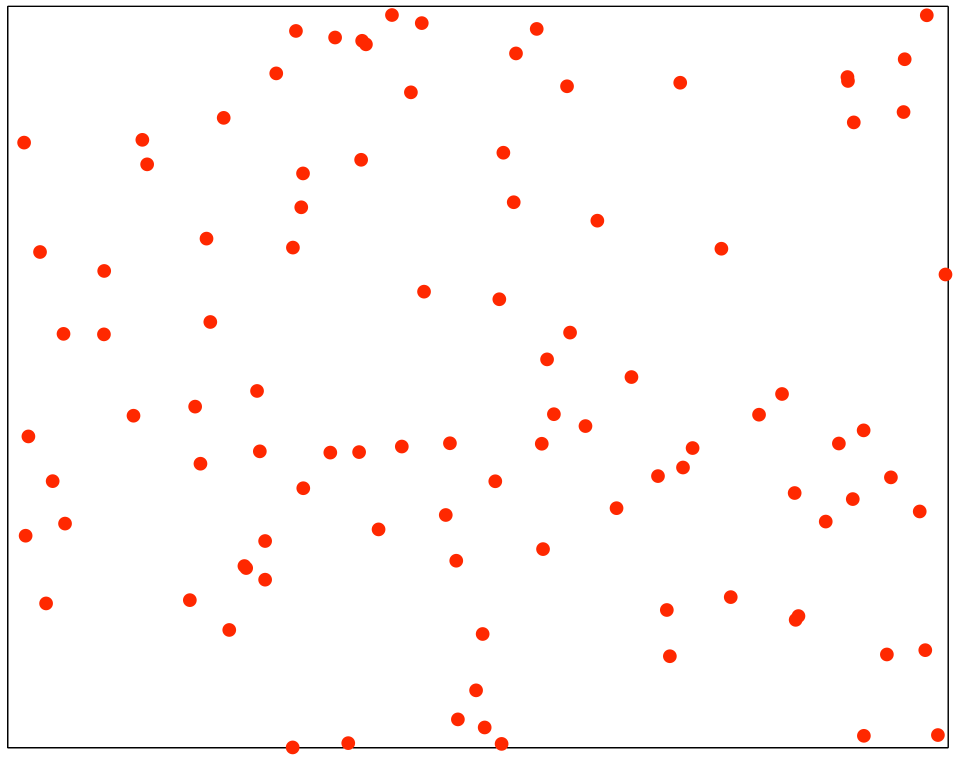
What is the true dimensionality of this data?



What is the true dimensionality of this data?



What is the true dimensionality of this data?

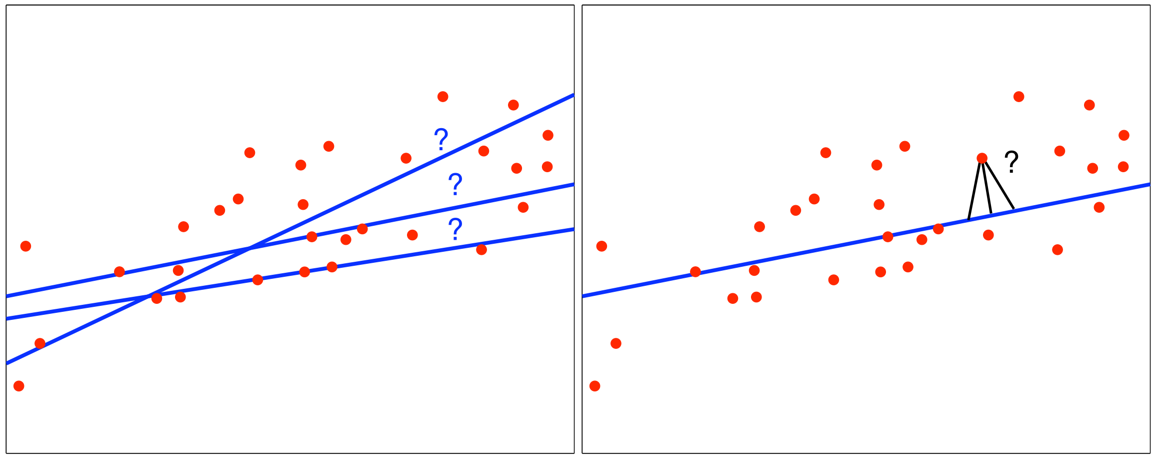


Remarks

- All dimensionality reduction techniques are based on an implicit assumption that the data lies along some *low-dimensional manifold*
- This is the case for the first three examples, which lie along a 1-dimensional manifold despite being plotted in 2D
- In the last example, the data has been generated randomly in 2D, so no dimensionality reduction is possible without losing information
- The first three cases are in increasing order of difficulty, from the point of view of existing techniques.

Simple Principal Component Analysis (PCA)

- Given: m instances, each being a length-n real vector.
- Suppose we want a 1-dimensional representation of that data, instead of n-dimensional.
- Specifically, we will:
  - Choose a line in $\mathbb{R}^n$ that "best represents" the data.
  - Assign each data object to a point along that line.



Reconstruction error

- Let the line be represented as $\mathbf{b} + \alpha\mathbf{v}$ for $\mathbf{b}, \mathbf{v} \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$.
  For convenience assume $\|\mathbf{v}\| = 1$.
- Each instance $\mathbf{x}_i$ is associated with a point on the line $\hat{\mathbf{x}}_i = \mathbf{b} + \alpha_i\mathbf{v}$.
- We want to choose $\mathbf{b}$, $\mathbf{v}$, and the α_i to minimize the total reconstruction error over all data points, measured using Euclidean distance:

$$R = \sum_{i=1}^{m} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

A constrained optimization problem!

$$\begin{array}{rl} \text{min} & \sum_{i=1}^{m} \|\mathbf{x}_i - (\mathbf{b} + \alpha_i \mathbf{v})\|^2 \\ \text{w.r.t.} & \mathbf{b}, \mathbf{v}, \alpha_i, i = 1, \ldots m \\ \text{s.t.} & \|\mathbf{v}\|^2 = 1 \end{array}$$

- This is a quadratic objective with quadratic constraint
- Suppose we fix a $\mathbf{v}$ satisfying the condition, and find the best $\mathbf{b}$ and α_i given this $\mathbf{v}$
- So, we solve:

$$\min R = \min_{\alpha, \mathbf{b}} \sum_{i=1}^{m} \|\mathbf{x}_i - (\mathbf{b} + \alpha_i \mathbf{v})\|^2$$

where R is the *reconstruction error*

Solving the optimization problem (II)

- We write the gradient of R wrt to α_i and set it to 0:

$$\frac{\partial R}{\partial \alpha_i} = 2\|\mathbf{v}\|^2\alpha_i - 2\mathbf{v}\mathbf{x}_i + 2\mathbf{b}\mathbf{v} = 0 \Rightarrow \alpha_i = \mathbf{v}\cdot(\mathbf{x}_i - \mathbf{b})$$

  where we take into account that $\|\mathbf{v}\|^2 = 1$.
- We write the gradient of R wrt $\mathbf{b}$ and set it to 0:

$$\nabla_{\mathbf{b}} R = 2m\mathbf{b} - 2\sum_{i=1}^{m}\mathbf{x}_i + 2\left(\sum_{i=1}^{m}\alpha_i\right)\mathbf{v} = 0 \tag{1}$$

- From above:

$$\sum_{i=1}^{m}\alpha_i = \sum_{i=1}^{m}\mathbf{v}^T(\mathbf{x}_i - \mathbf{b}) = \mathbf{v}^T\left(\sum_{i=1}^{m}\mathbf{x}_i - m\mathbf{b}\right) \tag{2}$$

Solving the optimization problem (III)

- By plugging (2) into (1) we get:

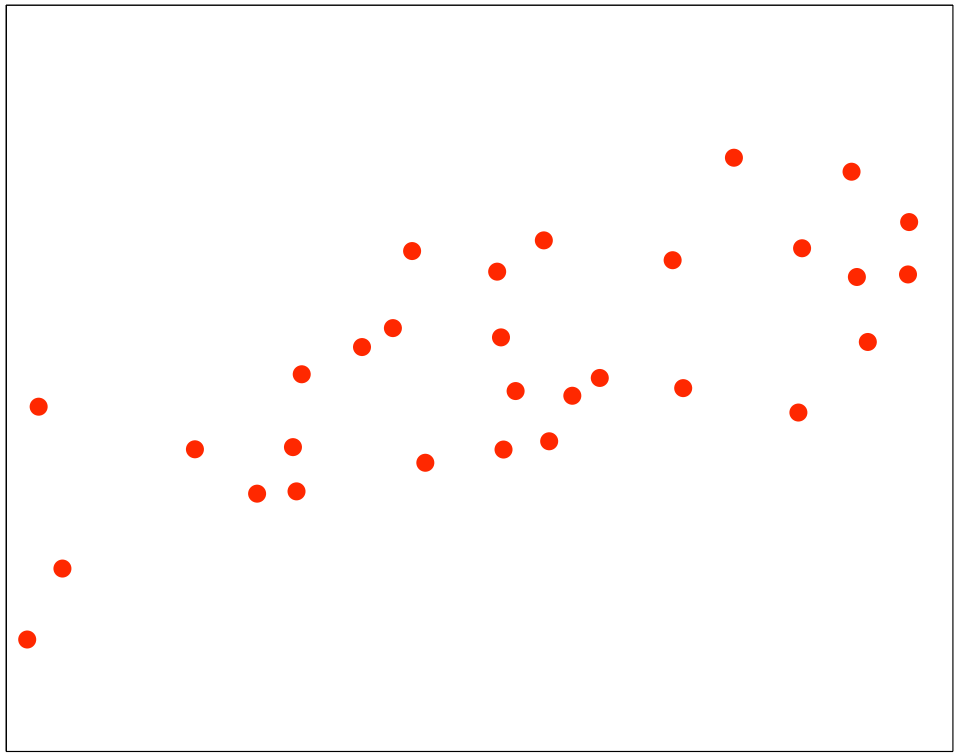
$$\mathbf{v}^T \left(\sum_{i=1}^{m} \mathbf{x}_i - m\mathbf{b} \right) \mathbf{v} = \left(\sum_{i=1}^{m} \mathbf{x}_i - m\mathbf{b} \right)$$

- This is satisfied when:

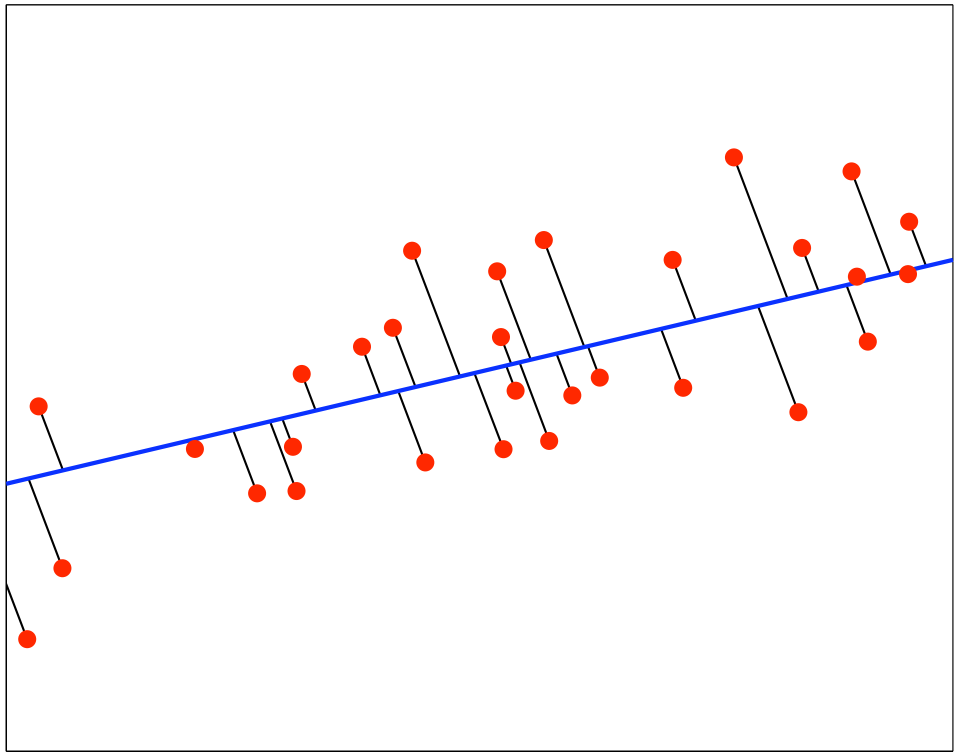
$$\sum_{i=1}^{m} \mathbf{x}_i - m\mathbf{b} = 0 \Rightarrow \mathbf{b} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_i$$

- This means that the line goes through the mean of the data
- By substituting α_i, we get: $\hat{\mathbf{x}}_i = \mathbf{b} + (\mathbf{v}^T(\mathbf{x}_i - \mathbf{b}))\mathbf{v}$
- This means that instances are projected orthogonally on the line to get the associated point.

Example data



Example with $\mathbf{v} \propto (1, 0.3)$



Finding the direction of the line

- Substituting $\alpha_i = \mathbf{v}^T(\mathbf{x}_i - \mathbf{b})$ into our optimization problem we obtain a new optimization problem:

$$\begin{array}{rl} \max_{\mathbf{v}} & \sum_{i=1}^{m} \mathbf{v}^T(\mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T\mathbf{v} \\ \text{s.t.} & \|\mathbf{v}\|^2 = 1 \end{array}$$

- The Lagrangian is:

$$L(\mathbf{v}, \lambda) = \sum_{i=1}^{m} \mathbf{v}^T(\mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T\mathbf{v} + \lambda - \lambda\|\mathbf{v}\|^2$$

- Let $S = \sum_{i=1}^{m}(\mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T$ be an n-by-n matrix, which we will call the *scatter matrix*
- The solution to the problem, obtained by setting $\nabla_{\mathbf{v}} L = 0$, is: $S\mathbf{v} = \lambda\mathbf{v}$.

Optimal choice of $\mathbf{v}$

- Recall: an *eigenvector* $\mathbf{u}$ of a matrix A satisfies $A\mathbf{u} = \lambda\mathbf{u}$, where $\lambda \in \mathbb{R}$ is the *eigenvalue*.
- Fact: the scatter matrix, S, has n non-negative eigenvalues and n orthogonal eigenvectors.
- The equation obtained for $\mathbf{v}$ tells us that it should be an eigenvector of S.
- The $\mathbf{v}$ that maximizes $\mathbf{v}^T S\mathbf{v}$ is the eigenvector of S with the largest eigenvalue

What is the scatter matrix

- S is an $n \times n$ matrix with

$$S(k,l) = \sum_{i=1}^{m} (\mathbf{x}_i(k) - \mathbf{b}(k))(\mathbf{x}_i(l) - \mathbf{b}(l))$$

- Hence, $S(k,l)$ is proportional to the *estimated covariance* between the kth and lth dimension in the data.

Recall: Covariance

- Covariance quantifies a *linear relationship* (if any) between two random variables X and Y.

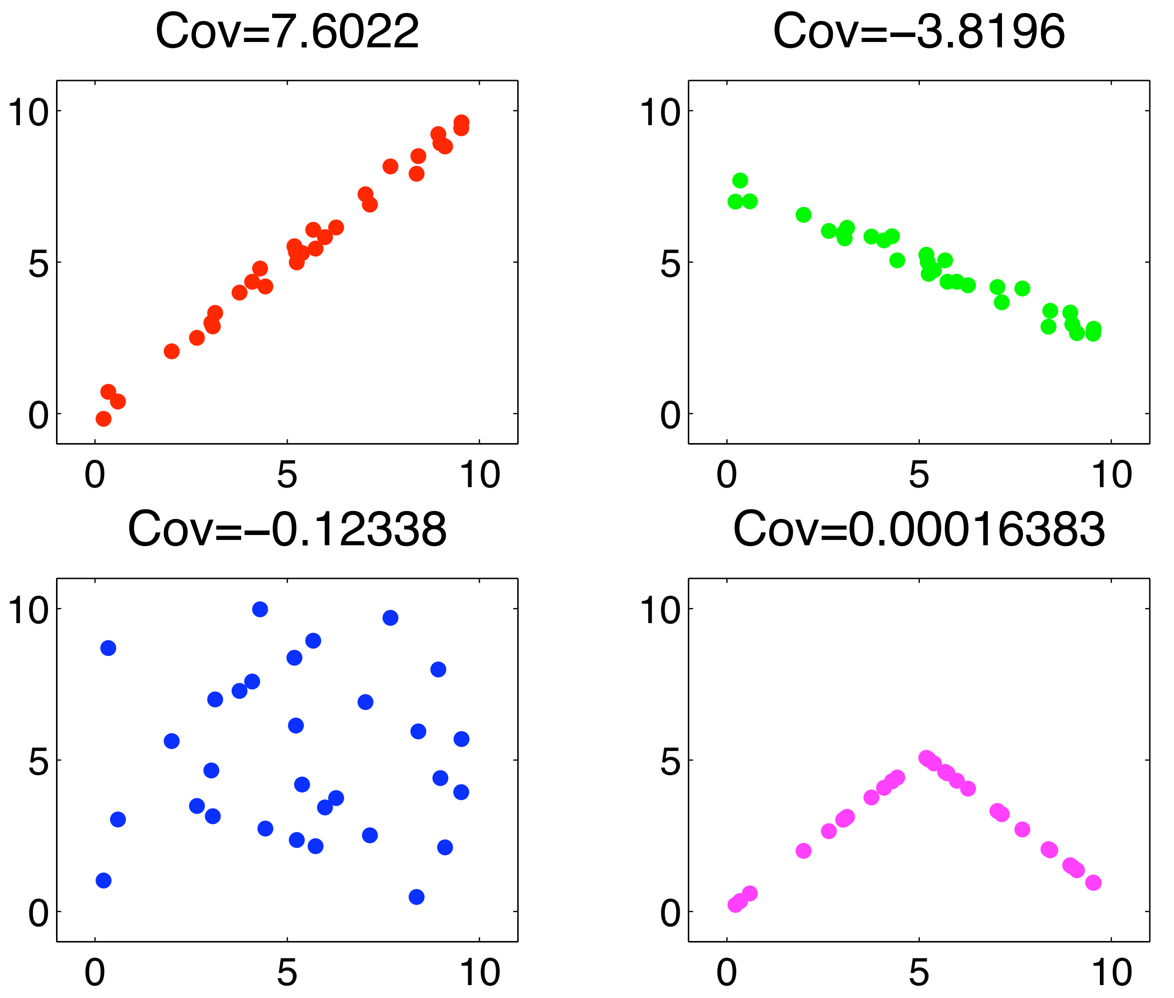
$$Cov(X,Y) = E\{(X - E(X))(Y - E(Y))\}$$

- Given m samples of X and Y, covariance can be estimated as

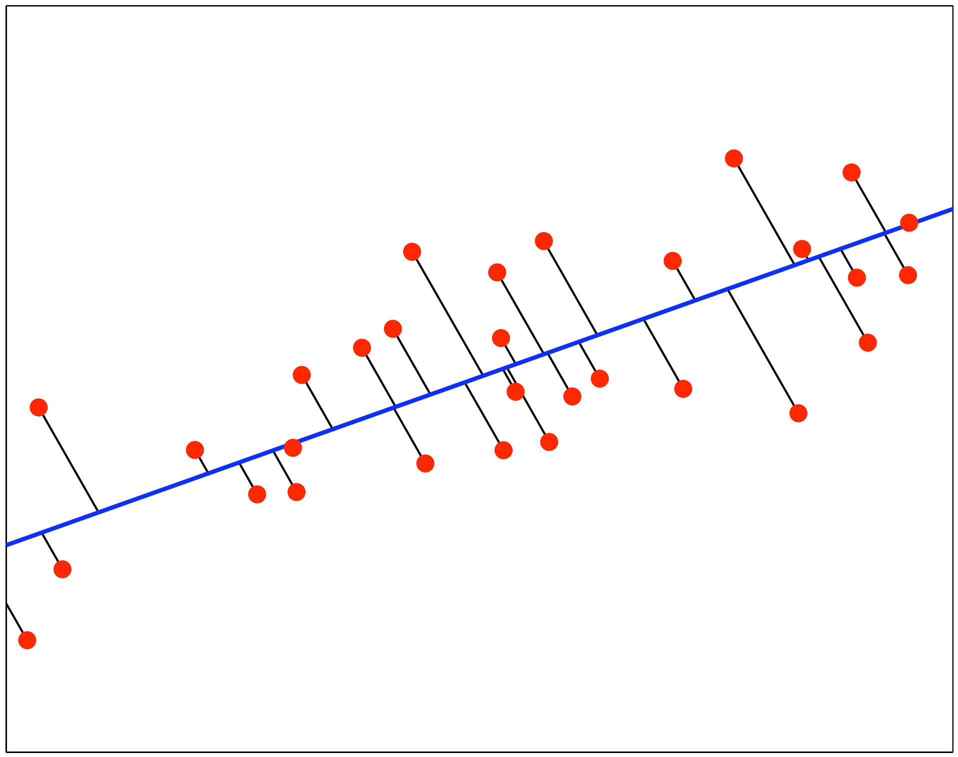
$$\frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_X)(y_i - \mu_Y)\ ,$$

  where $\mu_X = (1/m)\sum_{i=1}^{m} x_i$ and $\mu_Y = (1/m)\sum_{i=1}^{m} y_i$.
- Note: $Cov(X,X) = Var(X)$.

Covariance example



Example with optimal line: $\mathbf{b} = (0.54, 0.52)$, $\mathbf{v} \propto (1, 0.45)$



Remarks

- The line $\mathbf{b} + \alpha\mathbf{v}$ is the *first principal component*.
- The variance of the data along the line $\mathbf{b} + \alpha\mathbf{v}$ is as large as along any other line.
- $\mathbf{b}$, $\mathbf{v}$, and the α_i can be computed easily in polynomial time.

Reduction to d dimensions

- More generally, we can create a d-dimensional representation of our data by projecting the instances onto a hyperplane $\mathbf{b} + \alpha^1\mathbf{v}_1 + \ldots + \alpha^d\mathbf{v}_d$.
- If we assume the $\mathbf{v}_j$ are of unit length and orthogonal, then the optimal choices are:
  - $\mathbf{b}$ is the mean of the data (as before)
  - The $\mathbf{v}_j$ are orthogonal eigenvectors of S corresponding to its d largest eigenvalues.
  - Each instance is projected orthogonally on the hyperplane.

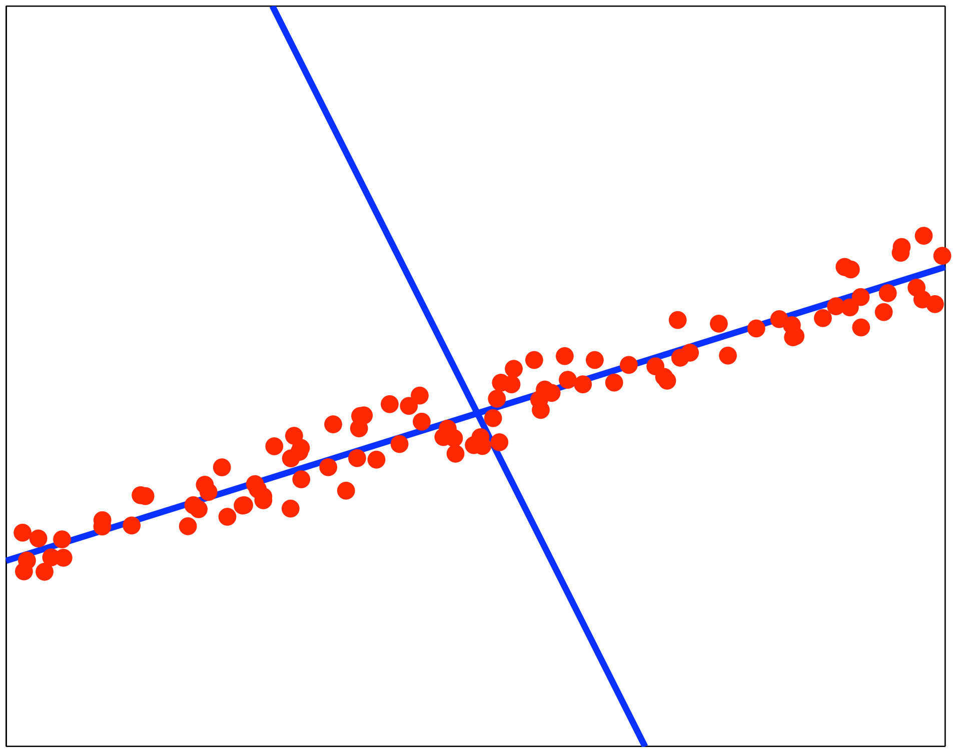
Remarks

- $\mathbf{b}$, the eigenvalues, the $\mathbf{v}_j$, and the projections of the instances can all be computing in polynomial time.
- The magnitude of the j^{th}-largest eigenvalue, λ_j, tells you how much variability in the data is captured by the j^{th} principal component
- So you have feedback on how to choose d!
- When the eigenvalues are sorted in decreasing order, the proportion of the variance captured by the first d components is:

$$\frac{\lambda_1 + \cdots + \lambda_d}{\lambda_1 + \cdots + \lambda_d + \lambda_{d+1} + \cdots + \lambda_n}$$

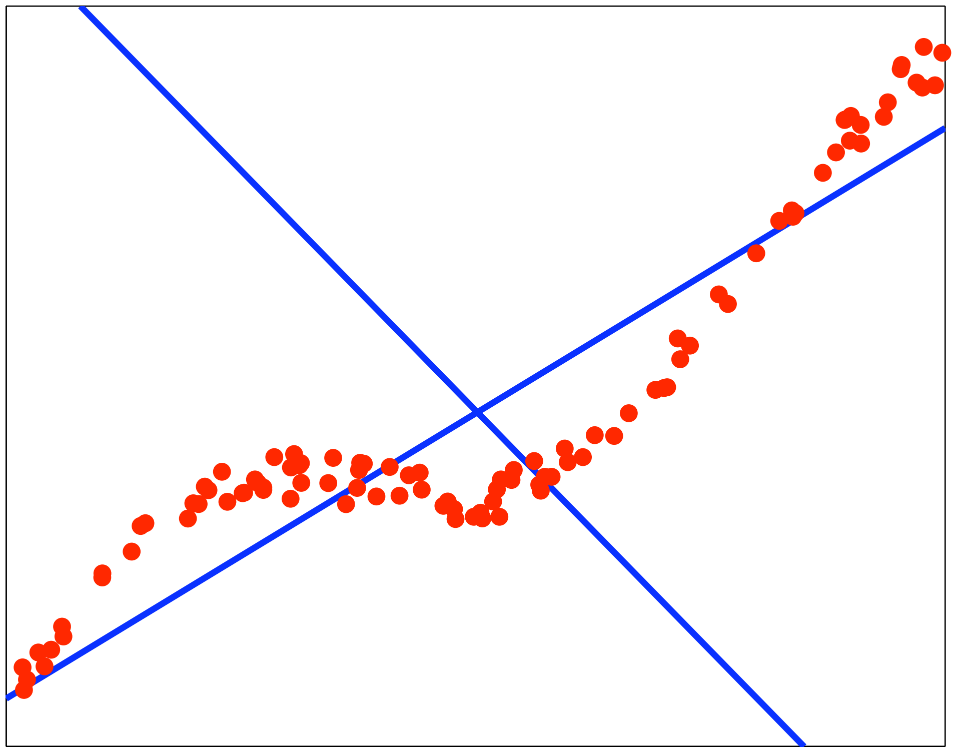
- So if a "big" drop occurs in the eigenvalues at some point, that suggests a good dimension cutoff

Example: $\lambda_1 = 0.0938, \lambda_2 = 0.0007$



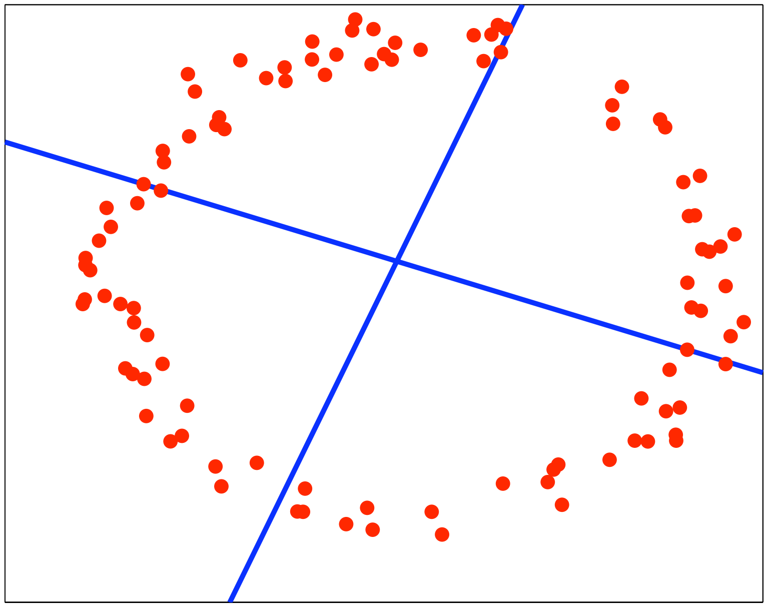
The first eigenvalue accounts for most variance, so the dimensionality is 1

Example: $\lambda_1 = 0.1260, \lambda_2 = 0.0054$



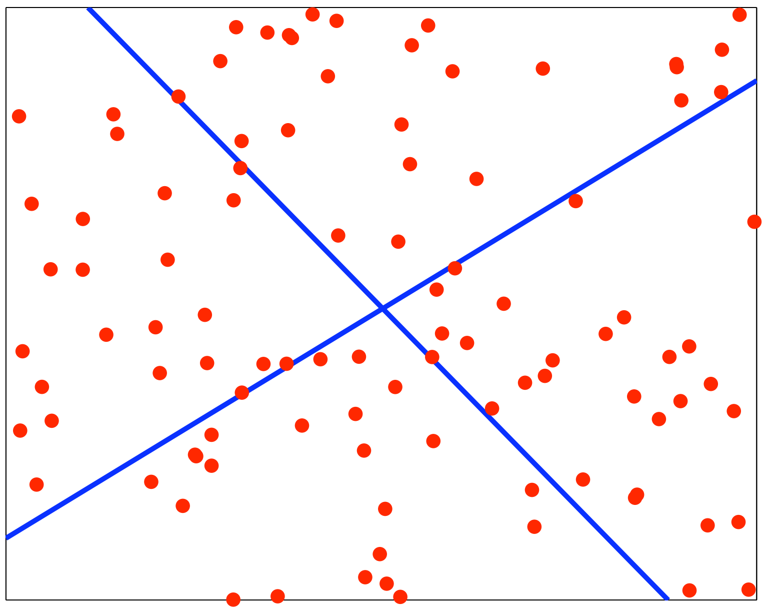
The first eigenvalue accounts for most variance, so the dimensionality is 1 (despite some non-linear structure in the data)

Example: $\lambda_1 = 0.0884, \lambda_2 = 0.0725$



- Each eigenvalue accounts for about half the variance, so the PCA-suggested dimension is 2
- Note that this is the *linear* dimension
- The true "non-linear" dimension of the data is 1 (using polar coordinates)

Example: $\lambda_1 = 0.0881, \lambda_2 = 0.0769$

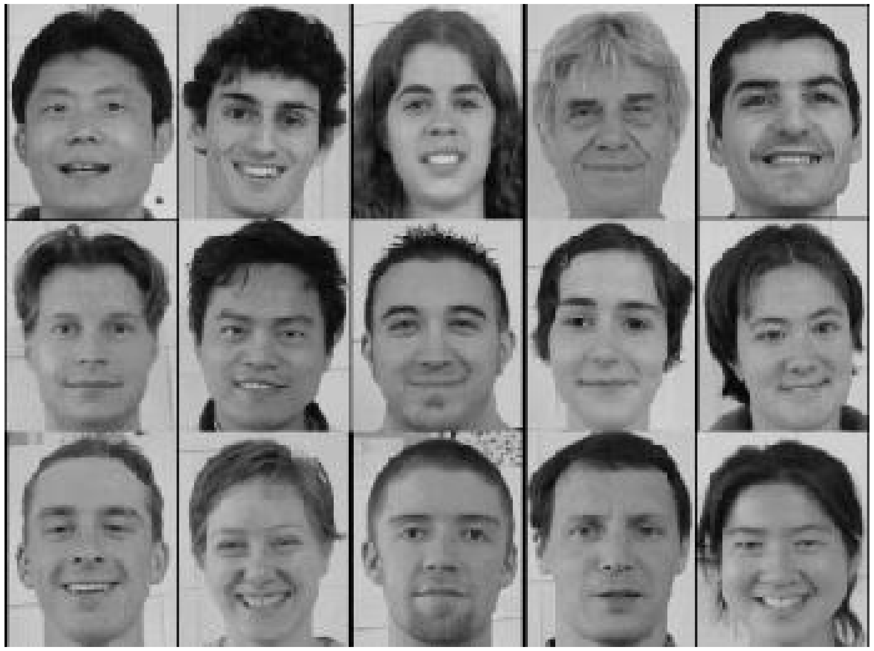


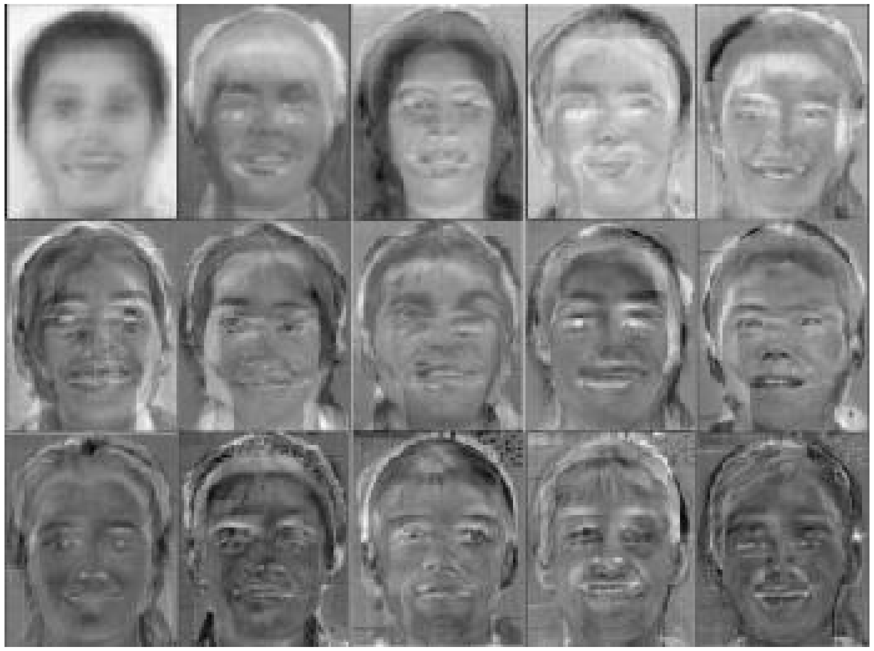
- Each eigenvalue accounts for about half the variance, so the PCA-suggested dimension is 2
- In this case, the non-linear dimension is also 2 (data is fully random)
- Note that *PCA cannot distinguish non-linear structure from no structure*
- This case and the previous one yield a very similar PCA analysis

Remarks

- Outliers have a big effect on the covariance matrix, so they can affect the eigenvectors quite a bit
- A simple examination of the pairwise distances between instances can help discard points that are very far away (for the purpose of PCA)
- If the variances in the original dimensions vary considerably, they can "muddle" the true correlations. There are two solutions:
  - Work with the correlation of the original data, instead of covariance matrix (which provides one type of normalization
  - Normalize the input dimensions individually (possibly based on domain knowledge) before PCA
- PCA is most often performed using Singular Value Decomposition (SVD)
- In certain cases, the eigenvectors are meaningful; e.g. in vision, they can be displayed as images ("eigenfaces")

Eigenfaces example



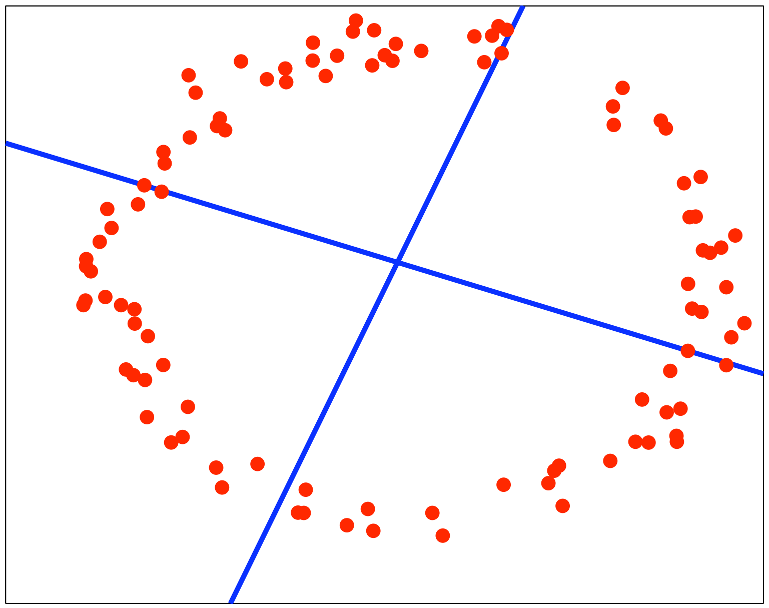


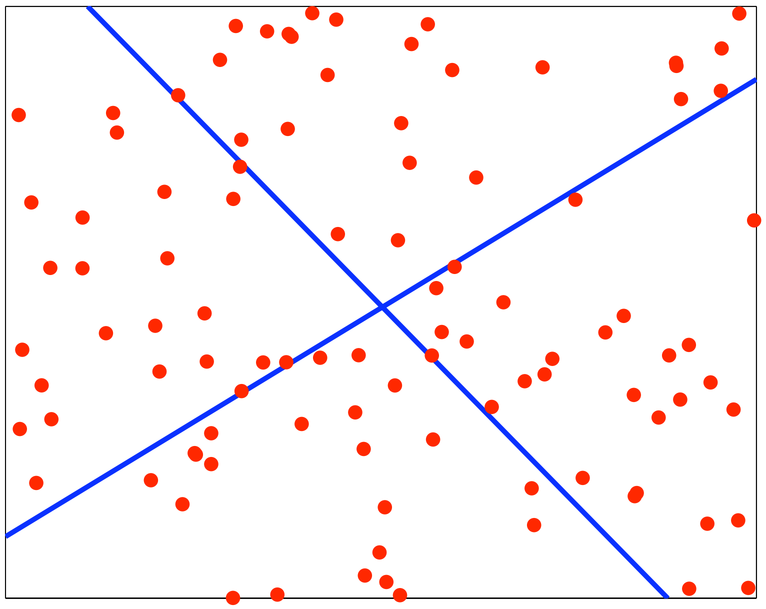
- A set of faces on the left and the corresponding eigenfaces (principal components) on the right
- Note that faces have to be centred and scaled ahead of time
- The components are in the same space as the instances (images) and can be used to reconstruct the images

Uses of PCA

- Pre-processing for a supervised learning algorithm, e.g. for image data, robotic sensor data
- Used with great success in image and speech processing
- Visualization
- Exploratory data analysis
- Removing the linear component of a signal (before fancier non-linear models are applied)

Difficult example





- PCA will make no difference between these examples, because the structure on the left is not linear
- Are there ways to find non-linear, low-dimensional manifolds?

Making PCA non-linear

- Suppose that instead of using the points $\mathbf{x}_i$ as is, we wanted to go to some different *feature space* $\phi(\mathbf{x}_i) \in \mathbb{R}^N$
- E.g. using polar coordinates instead of cartesian coordinates would help us deal with the circle
- In the higher dimensional space, we can then do PCA
- The result will be non-linear in the original data space!
- Similar idea to support vector machines

PCA in feature space (I)

- Suppose for the moment that the mean of the data in feature space is 0, so: $\sum_{i=1}^{m} \phi(\mathbf{x}_i) = 0$
- The covariance matrix is:

$$\mathbf{C} = \frac{1}{m} \sum_{i=1}^{m} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T$$

- The eigenvectors are:

$$\mathbf{C}\mathbf{v}_j = \lambda_j \mathbf{v}_j, j = 1, \ldots N$$

- We want to avoid explicitly going to feature space - instead we want to work with *kernels*:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_k)$$

PCA in feature space (II)

- Re-write the PCA equation:

$$\frac{1}{m}\sum_{i=1}^{m}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\mathbf{v}_j = \lambda_j\mathbf{v}_j, j = 1, \dots N$$

- So the eigenvectors can be written as a linear combination for features:

$$\mathbf{v}_j = \sum_{i=1}^{m} a_{ji}\phi(\mathbf{x}_i)$$

- Finding the eigenvectors is equivalent to finding the coefficients $a_{ji}, j = 1, \dots N, i = 1, \dots m$

PCA in feature space (III)

- By substituting this back into the equation we get:

$$\frac{1}{m}\sum_{i=1}^{m}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\left(\sum_{l=1}^{m}a_{jl}\phi(\mathbf{x}_l)\right)=\lambda_j\sum_{l=1}^{m}a_{jl}\phi(\mathbf{x}_l)$$

- We can re-write this as:

$$\frac{1}{m}\sum_{i=1}^{m}\phi(\mathbf{x}_i)\left(\sum_{l=1}^{m}a_{jl}K(\mathbf{x}_i,\mathbf{x}_l)\right)=\lambda_j\sum_{l=1}^{m}a_{jl}\phi(\mathbf{x}_l)$$

- A small trick: multiply this by $\phi(\mathbf{x}_k)^T$ to the left:

$$\frac{1}{m}\sum_{i=1}^{m}\phi(\mathbf{x}_k)^T\phi(\mathbf{x}_i)\left(\sum_{l=1}^{m}a_{jl}K(\mathbf{x}_i,\mathbf{x}_l)\right)=\lambda_j\sum_{l=1}^{m}a_{jl}\phi(\mathbf{x}_k)^T\phi(\mathbf{x}_l)$$

PCA in feature space (IV)

- We plug in the kernel again:

$$\frac{1}{m}\sum_{i=1}^{m} K(\mathbf{x}_k,\mathbf{x}_i)\left(\sum_{l=1}^{m} a_{jl}K(\mathbf{x}_i,\mathbf{x}_l)\right) = \lambda_j \sum_{l=1}^{m} a_{jl}K(\mathbf{x}_k,\mathbf{x}_l), \forall j,k$$

- By rearranging we get: $\mathbf{K}^2\mathbf{a}_j = m\lambda_j\mathbf{K}\mathbf{a}_j$
- We can remove a factor of $\mathbf{K}$ from both sides of the matrix (this will only affect eigenvectors with eigenvalues 0, which will not be principle components anyway):

$$\mathbf{K}\mathbf{a}_j = m\lambda_j\mathbf{a}_j$$

PCA in feature space (V)

- We have a normalization condition for the $\mathbf{a}_j$ vectors:

$$\mathbf{v}_j^T\mathbf{v}_j = 1 \Rightarrow \sum_{k=1}^{m}\sum_{l=1}^{m} a_{jl}a_{jk}\phi(\mathbf{x}_l)^T\phi(\mathbf{x}_k) = 1 \Rightarrow \mathbf{a}_j^T\mathbf{K}\mathbf{a}_j = 1$$

- Plugging this into:

$$\mathbf{K}\mathbf{a}_j = m\lambda_j\mathbf{a}_j$$

  we get: $\lambda_j m\mathbf{a}_j^T\mathbf{a}_j = 1, \forall j$

- For a new point $\mathbf{x}$, its projection onto the principal components is:

$$\phi(\mathbf{x})^T\mathbf{v}_j = \sum_{i=1}^{m} a_{ji}\phi(\mathbf{x})^T\phi(\mathbf{x}_i) = \sum_{i=1}^{m} a_{ji}K(\mathbf{x}, \mathbf{x}_i)$$

Normalizing the feature space

- In general, the features $\phi(\mathbf{x}_i)$ may not have mean 0
- We want to work with:

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

- The corresponding kernel matrix entries are given by:

$$\tilde{K}(\mathbf{x}_k, \mathbf{x}_l) = \tilde{\phi}(\mathbf{x}_l)^T\tilde{\phi}(\mathbf{x}_j)$$

- After some algebra, we get:

$$\tilde{\mathbf{K}} = \mathbf{K} - 2\mathbf{1}_{1/m}\mathbf{K} + \mathbf{1}_{1/m}\mathbf{K}\mathbf{1}_{1/m}$$

where $\mathbf{1}_{1/m}$ is the matrix with all elements equal to $1/m$

Summary of kernel PCA

1. Pick a kernel
2. Construct the normalized kernel matrix $\tilde{\mathbf{K}}$ of the data (this will be of dimension $m \times m$)
3. Find the eigenvalues and eigenvectors of this matrix λ_j, $\mathbf{a}_j$
4. For any data point (new or old), we can represent it as the following set of features:
$$y_j = \sum_{i=1}^{m} a_{ji} K(\mathbf{x}, \mathbf{x}_i), j = 1, \ldots m$$
5. We can limit the number of components to $k < m$ for a more compact representation (by picking the $\mathbf{a}$'s corresponding to the highest eigenvalues)

Representation obtained by kernel PCA

- Each y_j is the coordinate of $\phi(\mathbf{x})$ along one of the feature space axes $\mathbf{v}_j$
- Remember that $\mathbf{v}_j = \sum_{i=1}^{m} a_{ji}\phi(\mathbf{x}_i)$ (the sum goes to k if $k < m$)
- Since $\mathbf{v}_j$ are orthogonal, the projection of $\phi(\mathbf{x})$ onto the space spanned by them is:

$$\Pi\phi(\mathbf{x}) = \sum_{j=1}^{m} y_j \mathbf{v}_j = \sum_{j=1}^{m} y_j \sum_{i=1}^{m} a_{ji}\phi(\mathbf{x}_i)$$

  (again, sums go to k if $k < m$)
- The *reconstruction error in feature space* can be evaluated as:

$$\|\phi(\mathbf{x}) - \Pi\phi(\mathbf{x})\|^2$$

  This can be re-written by expanding the norm; we obtain dot-products which can all be replaced by kernels
- Note that the error will be 0 on the training data if enough $\mathbf{v}_j$ are retained

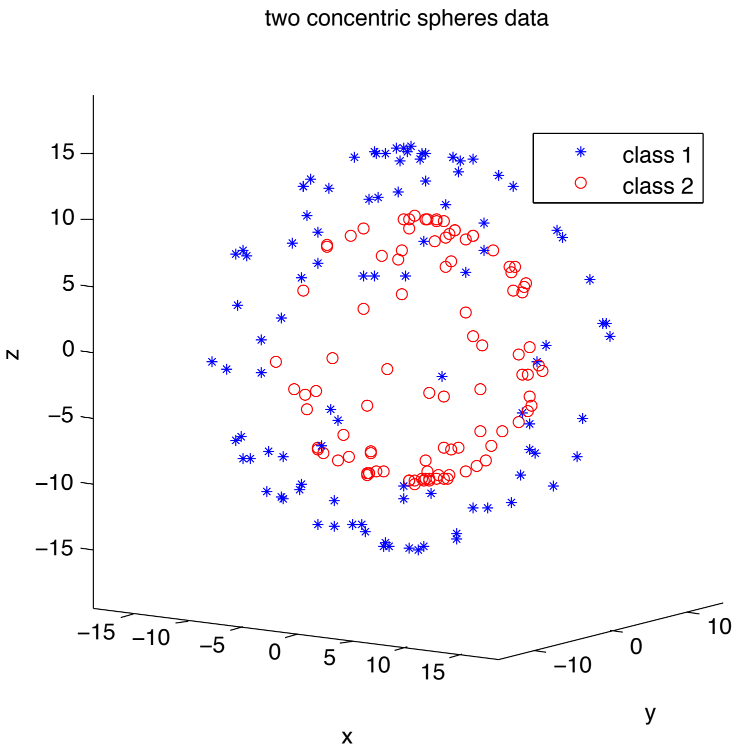
Alternative reconstruction error measures

- An alternative way of measuring performance is by looking at how well kernel PCA preserves distances between data points
- In this case, the Euclidian distance in kernel space between points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$, d_{ij}, is:

$$\|\phi\mathbf{x}_i) - \phi(\mathbf{x}_j)\| = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)$$

- The distance $\hat{d}_i j$ between the projected points in kernel space is defined as above, but with $\phi(\mathbf{x}_i)$ replaced by $\Pi\phi(\mathbf{x}_i)$.
- The average of $d_{ij} - \hat{d}_{ij}$ over all pairs of points is a measure of reconstruction error
- Note that reconstruction error in the original space of the $\mathbf{x}_i$ is very difficult to compute, because it requires taking $\Pi\phi(\mathbf{x})$ and finding its pre-image in the original feature space, which is not always feasible (though approximations exist)

Example: Two concentric spheres

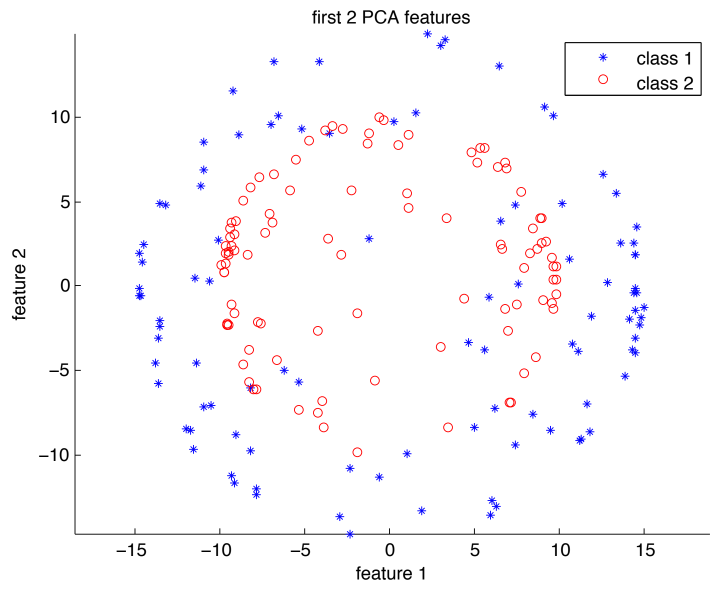


[1]

- Colours are used for clarity in the picture, but the data is presented unlabelled
- We want to project form 3D to 2D

[1]Wang, 2012

Example: Two concentric spheres - PCA

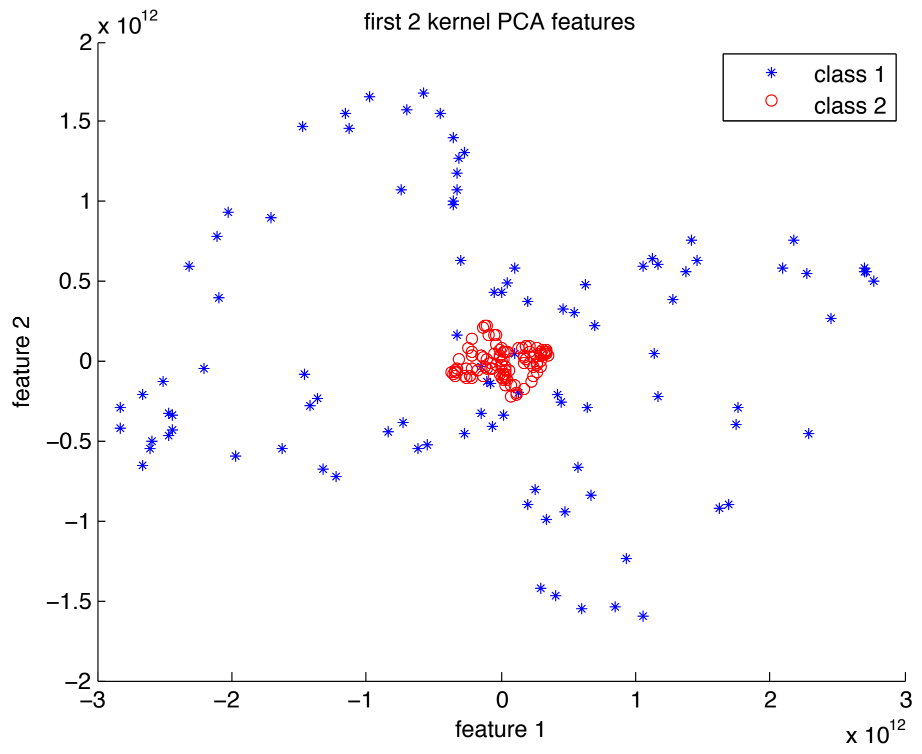


[2]

Note that PCA is unable to separate the points from the two spheres

[2]Wang, 2012

Example: Kernel PCA with Polynomial Kernel ($d = 5$)

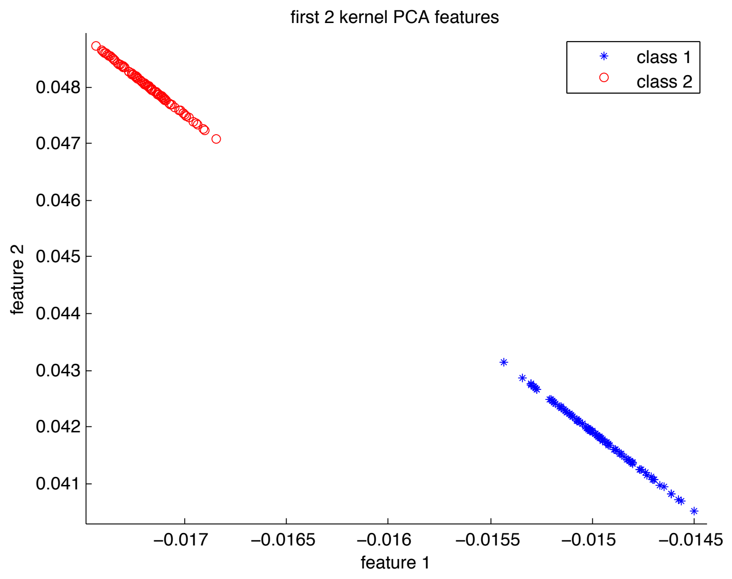


[3]

- Points from one sphere are much closer together, the others are scattered
- The projected data is not linearly separable

[3] Wang, 2012

Example: Kernel PCA with Gaussian Kernel ($\sigma = 20$)



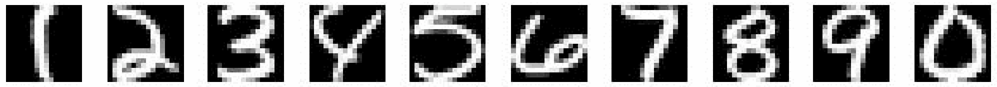
[4]

- Points from the two spheres are really well separated
- Note that the choice of parameter for the kernel matters!
- Validation can be used to determine good kernel parameter values

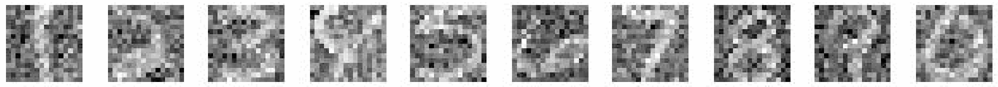
[4]Wang, 2012

Example: De-noising images

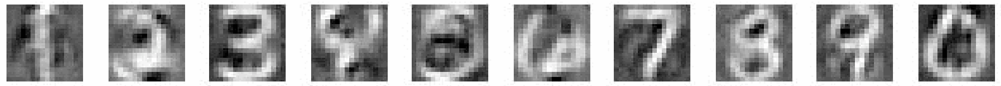
Original data



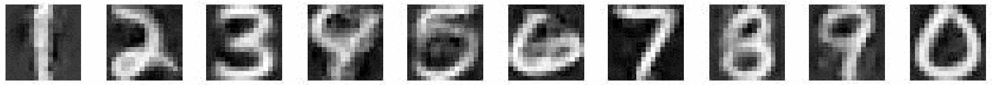
Data corrupted with Gaussian noise



Result after linear PCA



Result after kernel PCA, Gaussian kernel



PCA vs Kernel PCA

- Kernel PCA can give a good re-encoding of the data when it lies along a non-linear manifold
- The kernel matrix is $m \times m$, so kernel PCA will have difficulties if we have lots of data points
- In this case, we may need to use dictionary methods to pick a subset of the data
- For general kernels, we may not be able to easily visualizethe image of a point in the input space, though visualization still works for simple kernels

Locally Linear Embedding

- $\mathbf{x}_1, \cdots, \mathbf{x}_m \in \mathbb{R}^n$ lies on a k-dimensional manifold.

$\Rightarrow$ Each point and its neighbors lie close to a *locally linear* patch of the manifold.

- We try to reconstruct each point from its neighbors:

$$\min_{\mathbf{W}} \sum_i \|\mathbf{x}_i - \sum_j \mathbf{W}_{i,j}\mathbf{x}_j\|^2$$

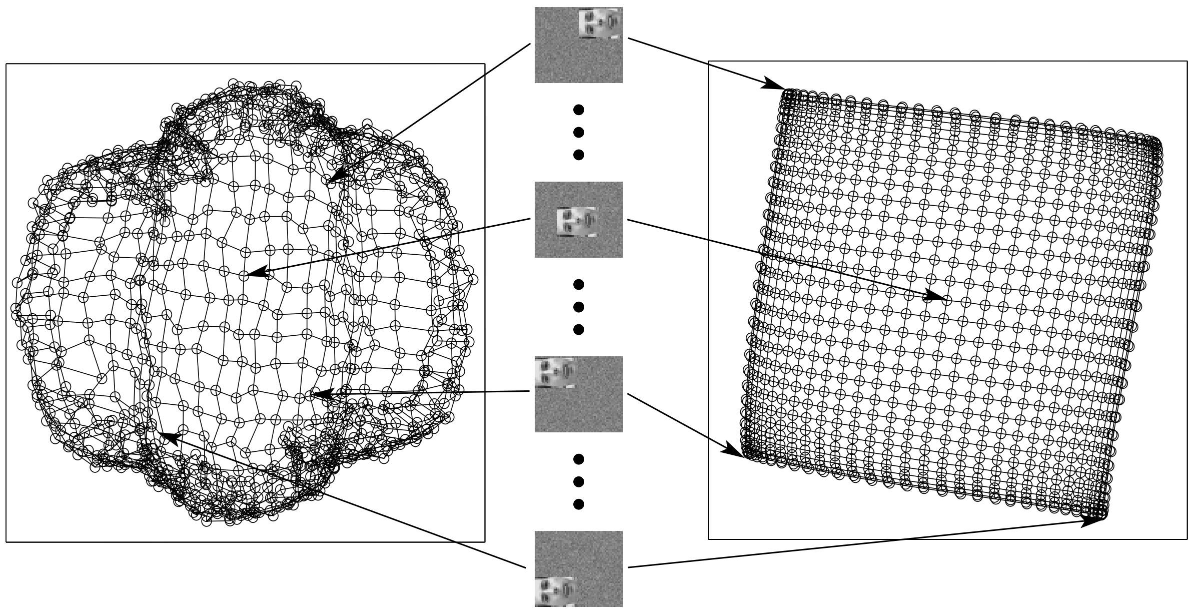
s.t. $\mathbf{W}\mathbf{1} = \mathbf{1}$ and $\mathbf{W}_{i,j} = 0$ if $\mathbf{x}_j \notin neighbors(\mathbf{x}_i)$

$\Rightarrow$ For each point the weights are invariant to rotation, scaling and translations: the weights $\mathbf{W}_{i,j}$ capture intrinsic geometric properties of each neighborhood.

- These local properties of each neighborhood should be preserved by the embedding:

$$\min_{\mathbf{z}_1,\dots,\mathbf{z}_m \in \mathbb{R}^k} \sum_i \|\mathbf{z}_i - \sum_j \mathbf{W}_{i,j}\mathbf{z}_j\|^2$$

PCA vs Locally Linear Embedding



[Saul, L. K., & Roweis, S. T. (2000). An introduction to locally linear embedding.]

Multi-dimensional scaling

- Input:
  - An $m \times m$ dissimilarity matrix d, where $d(i,j)$ is the distance between instances $\mathbf{x}_i$ and $\mathbf{x}_j$
  - Desired dimension k of the embedding.
- Output:
  - Coordinates $\mathbf{z}_i \in \mathbb{R}^k$ for each instance i that minimize a "stress" function quantifying the mismatch between distances as given by d and distances of the data representation in $\mathbb{R}^k$.

Stress functions

- Common stress functions include:
  - The least-squares or Kruskal-Shephard criterion:

$$\sum_{i=1}^{m} \sum_{j \neq i} (d(i,j) - \|\mathbf{z}_i - \mathbf{z}_j\|)^2$$

  - The Sammon mapping:

$$\sum_{i=1}^{m} \sum_{j \neq i} \frac{(d(i,j) - \|\mathbf{z}_i - \mathbf{z}_j\|)^2}{d(i,j)} \ ,$$

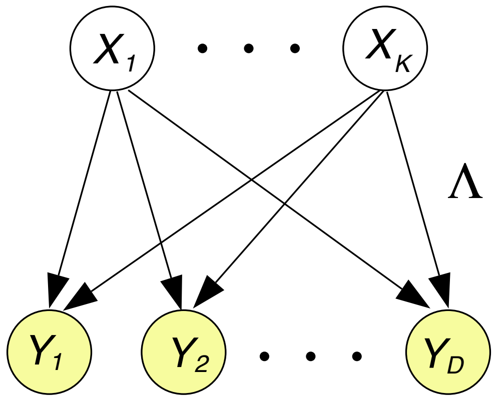
    which emphasizes getting small distances correct.

- Gradient-based optimization is usually used to find $\mathbf{z}_i$

Other dimensionality reduction methods

- Independent component analysis (ICA)
- More generally: factor analysis
- Local linear embeddings (LLE)
- Neighborhood component analysis (NCA)
- ...
- Some methods do dimensionality reduction jointly with a supervised learning task, or a set of such tasks

A generalizing perspective

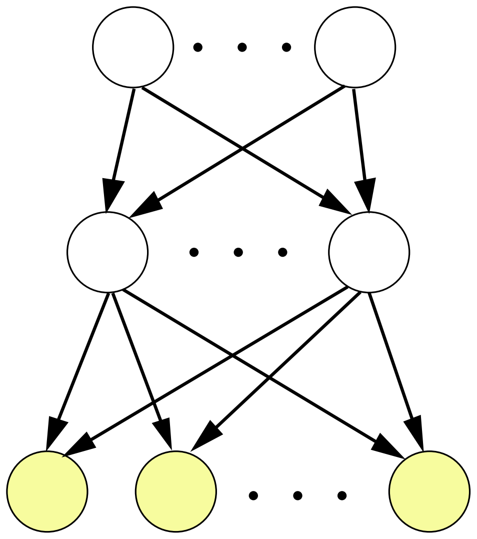


[5]

- Let Y be observed data and X be *hidden (latent) variables* or *factors* that generate the data
- The goal is to find how many such variables there are, and the model through which they generate the data
- E.g. Mixture models: K hidden variables, Gaussian conditional distributions
- E.g. PCA: K hidden variables, Gaussian models
- E.g. ICA: K hidden variables, non-Gaussian models

[5] Roweis and Gharamani, 1999

Graphical models

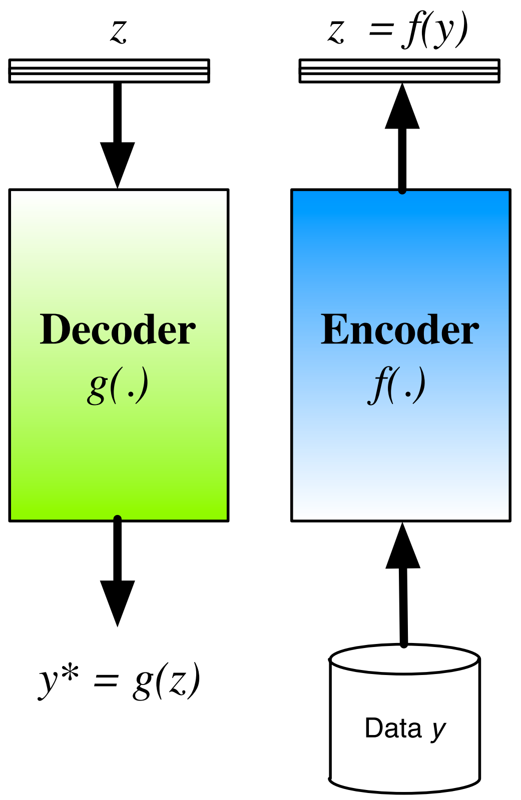


- More generally, the data (yellow circles) can be generated by a more complex structure.
- We can model all variables and their interactions using a graph structure
- Local probabilistic models describe how neighbours influence each other
- The overall model represents a joint probability distribution over all variables (observed and latent)

More generally: Autoencoders

- We have some data and try to learn a latent variable space that explains it
- The goal is to minimize reconstruction error
- In PCA, we used squared loss - this indicates an implicit Gaussian assumption
- More generally, from data bfy we obtain a mapping $\mathbf{z}$, then we can use an *inverse mapping* g to go back from $\mathbf{z}$ to $\mathbf{y}$
- We want to maximize the likelihood of the data

More generally: Autoencoders



$$\mathcal{L} = -\log p(y|g(z))$$
$$\mathcal{L} = \|y - g(f(y))\|_2^2$$

Two views of auto encoders

- We just implement functions for f, g (e.g. lots of sigmoids in layers) - this gives rise to deep auto encoders, trained by gradient descent
- We commit to full-blow probabilistic models, treating z as probabilistic random variable - this gives rise to variational auto encoders