Machine Learning 2

DS 4420 - Spring 2020

Dimensionality reduction I

Byron C Wallace

Machine Learning 2

DS 4420 - Spring 2020

Some slides today borrowing from:
Percy Liang (Stanford)

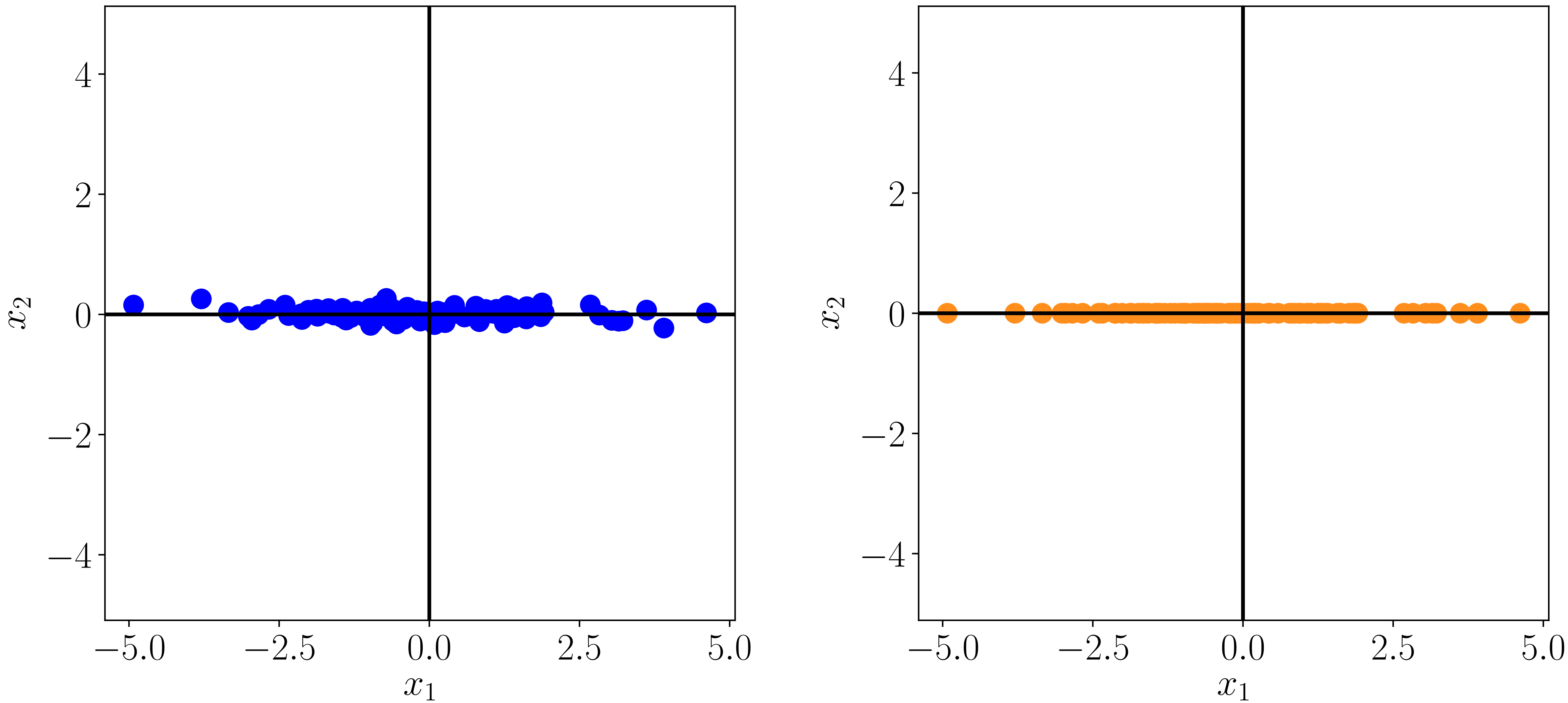
Other material from the MML book (Faisal and Ong)

Motivation

- We often want to work with *high dimensional* data (e.g., images). We also often have lots of it.
- This is computationally expensive to store and work with.

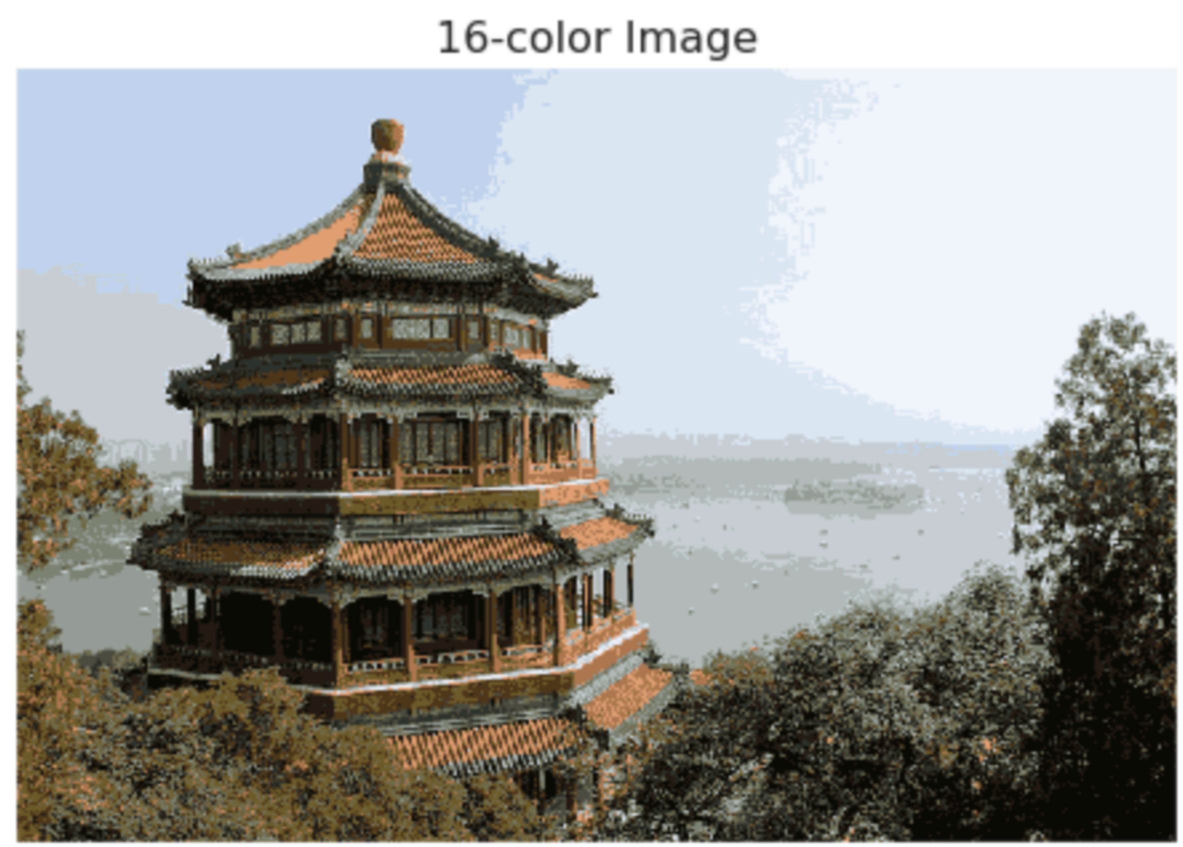
Dimensionality Reduction

Fundamental idea Exploit *redundancy* in the data; find *lower-dimensional* representation



Example (from lecture 5): Dimensionality reduction via *k*-means

Example (from lecture 5): Dimensionality reduction via *k*-means

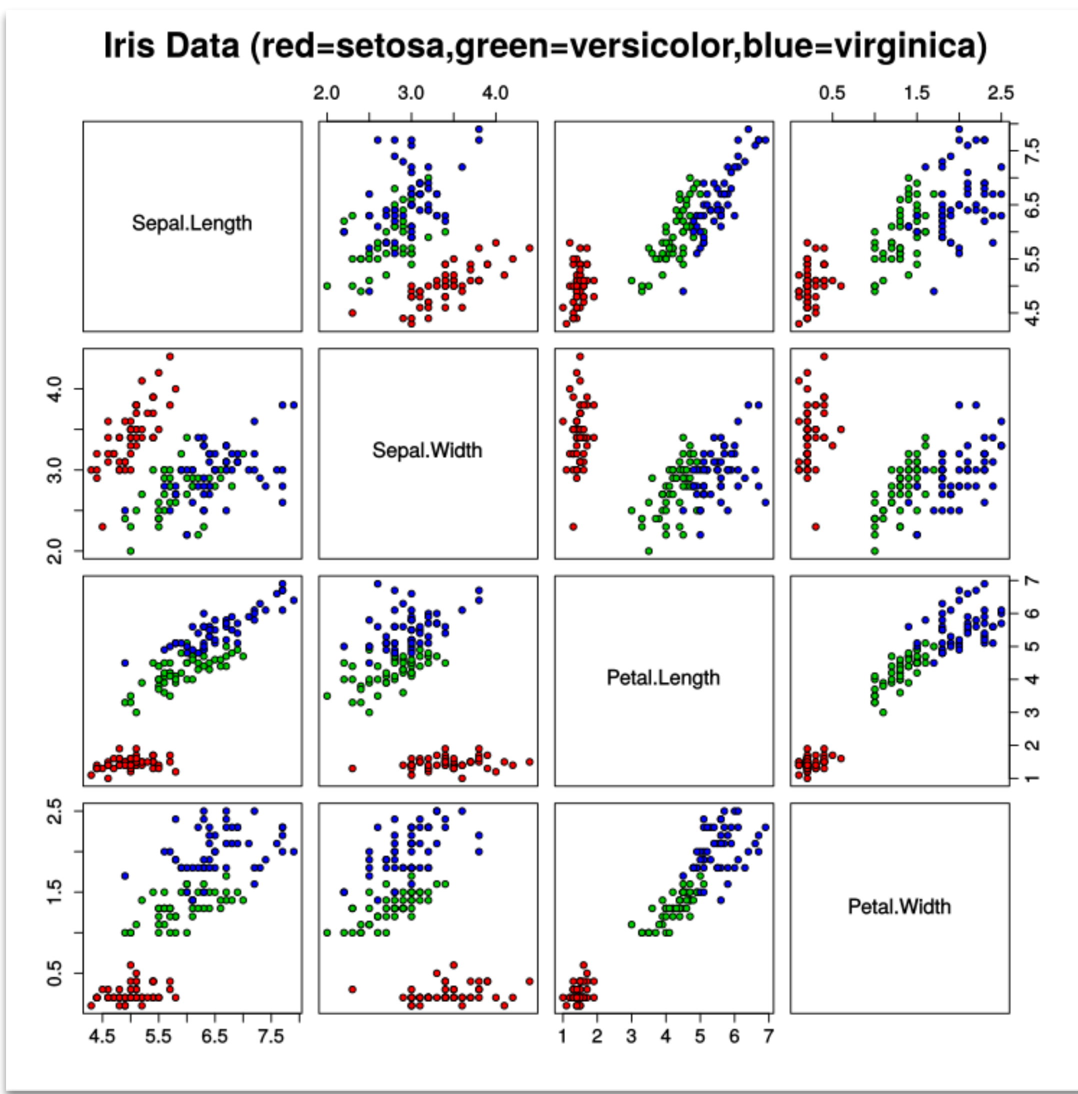


This highlights the natural connection between dimensionality reduction and *compression.*

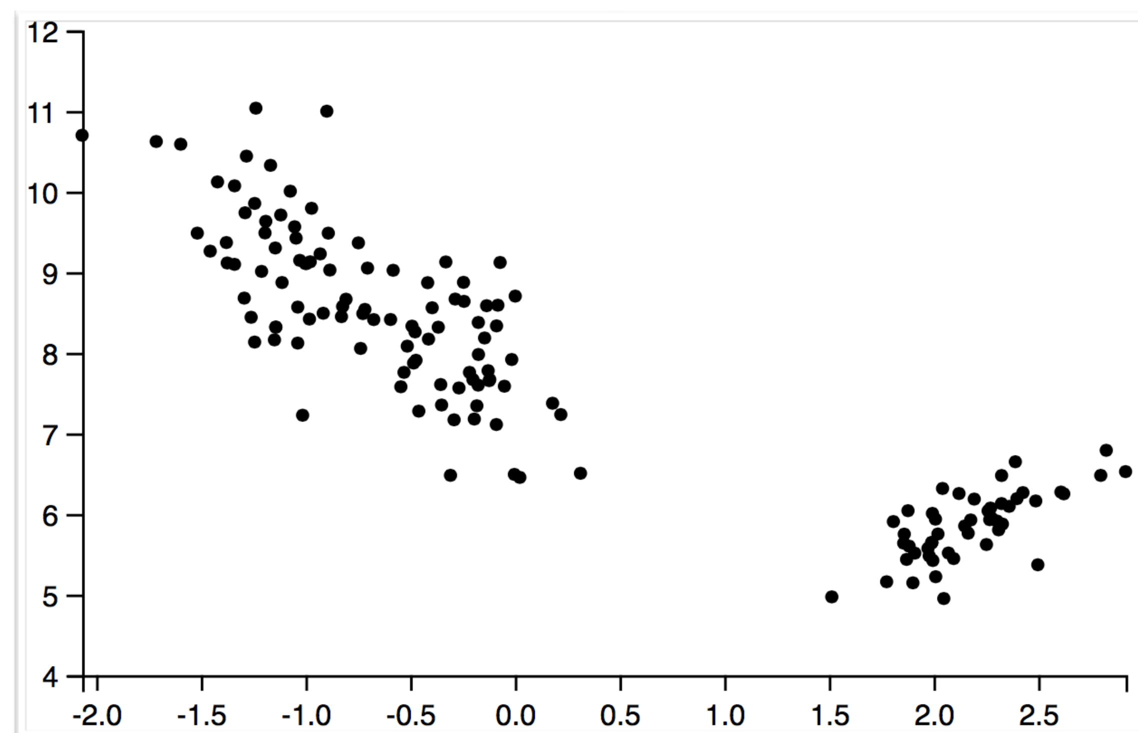
Dimensionality reduction

Goal: Map high dimensional data onto lower-dimensional data in a manner that preserves *distances/similarities*

Original Data (4 dims)



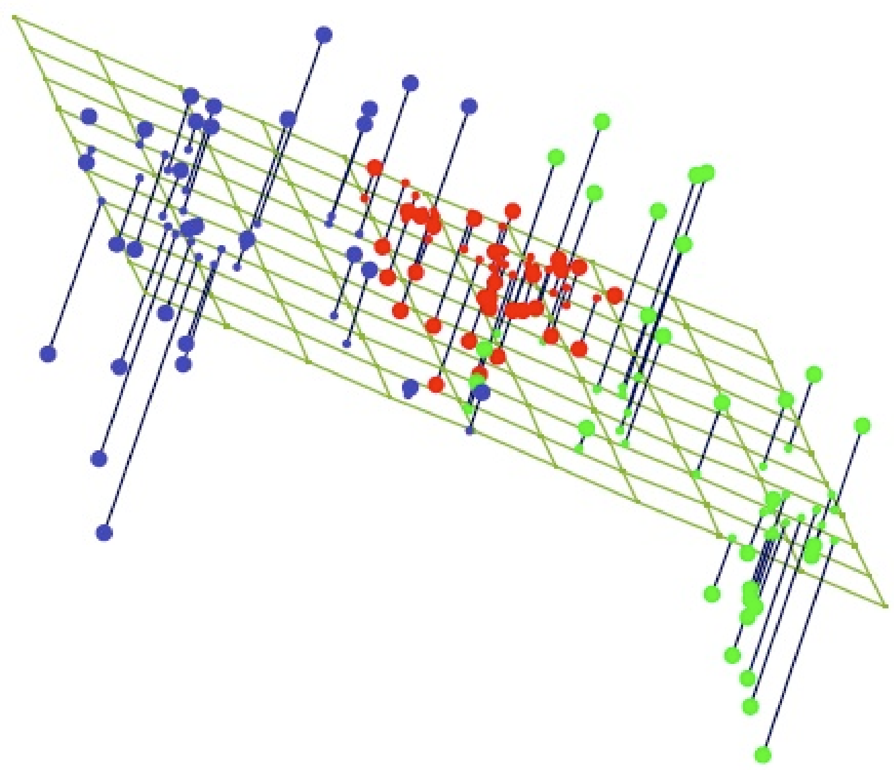
Projection with PCA (2 dims)



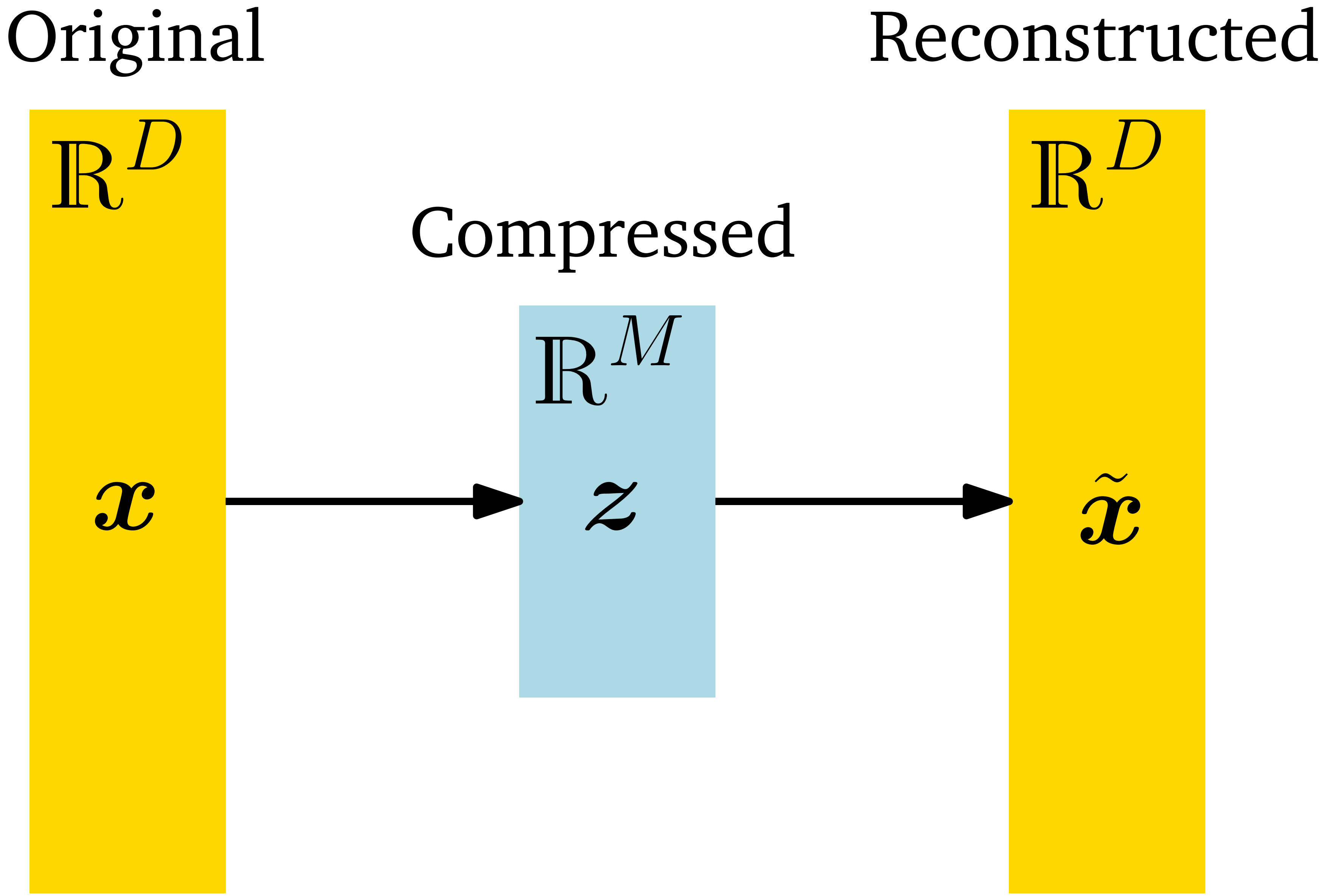
Objective: projection should "preserve" relative distances

Linear dimensionality reduction

Idea: Project high-dimensional vector onto a lower dimensional space



Linear dimensionality reduction



Objective

Key intuition:

$$\underbrace{\text{variance of data}}_{\text{fixed}} = \underbrace{\text{captured variance}}_{\text{want large}} + \underbrace{\text{reconstruction error}}_{\text{want small}}$$

Principal Component Analysis
(on board)

In Sum: Principal Component Analysis

Data

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots\cdots & \mathbf{x}_n \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times n}$$

Orthonormal Basis

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdot\cdot & \mathbf{u}_d \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times d}$$

Eigenvectors of Covariance

$$\mathbf{C} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j \mathbf{x}_j^\top = \frac{1}{n} \mathbf{X}\mathbf{X}^\top$$

$$\mathbf{C}\mathbf{u}_j = \lambda_j \mathbf{u}_j$$

Eigen-decomposition

$$\boldsymbol{C} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top$$

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ldots & \\ & & & \lambda_d \end{pmatrix}$$

Idea: Take top-*k* eigenvectors to maximize variance

Getting the eigenvalues, two ways

- Direct eigenvalue decomposition of the covariance matrix

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top$$

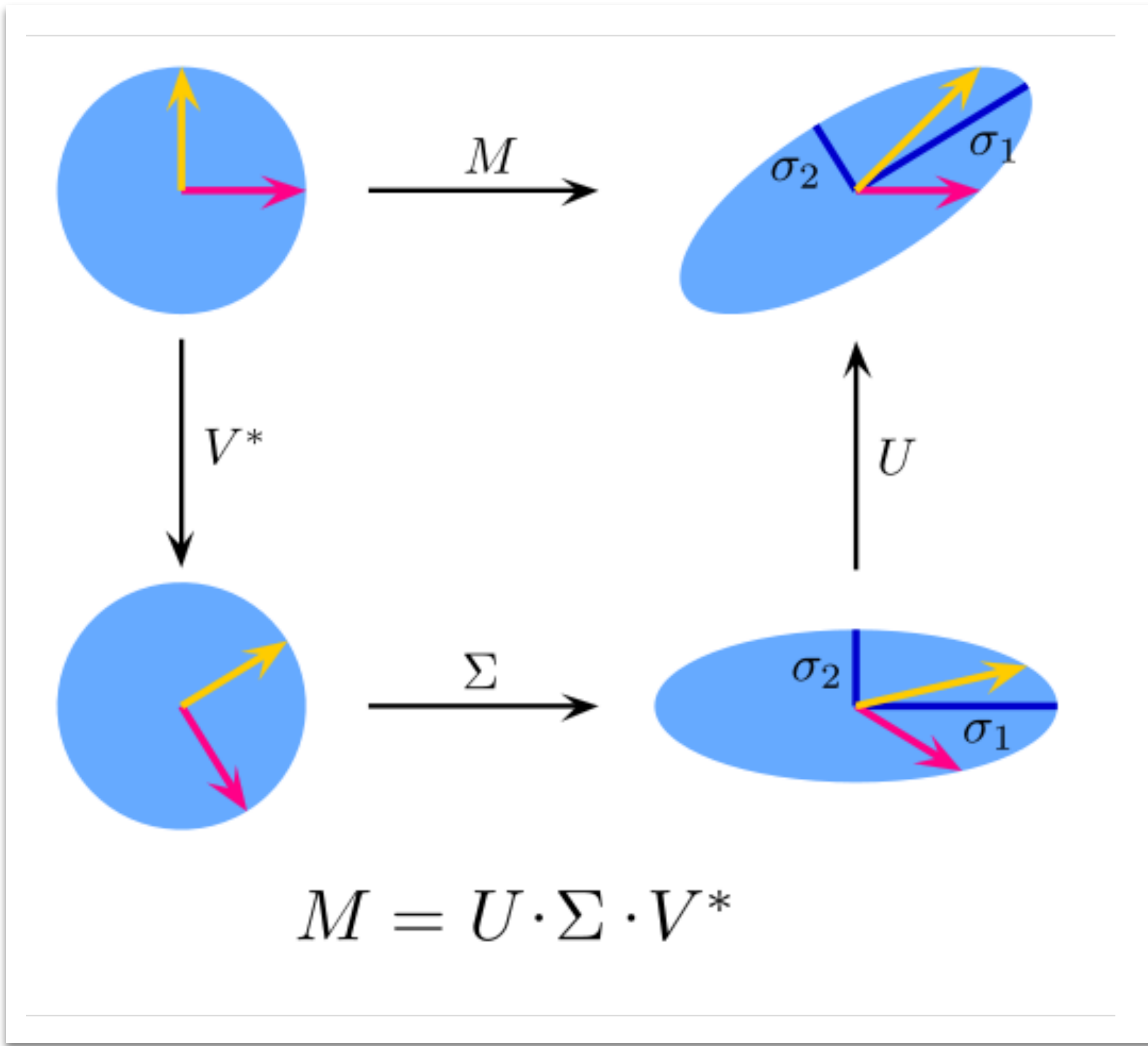
Getting the eigenvalues, two ways

- Direct eigenvalue decomposition of the covariance matrix

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top$$

- Singular Value Decomposition (SVD)

Singular Value Decomposition



Idea: Decompose the **d x n** matrix *X* into

1. A n x n basis *V* (unitary matrix)
2. A d x n matrix ***Σ*** (diagonal projection)
3. A d x d basis *U* (unitary matrix)

$$\mathbf{X} = \mathbf{U}_{d \times d} \Sigma_{d \times n} \mathbf{V}_{n \times n}^{\top}$$

SVD for PCA

$$\underbrace{\boldsymbol{X}}_{D\times N} = \underbrace{\boldsymbol{U}}_{D\times D}\underbrace{\boldsymbol{\Sigma}}_{D\times N}\underbrace{\boldsymbol{V}^\top}_{N\times N}$$

$$\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top = \frac{1}{N}\boldsymbol{U}\boldsymbol{\Sigma}\underbrace{\boldsymbol{V}^\top\boldsymbol{V}}_{=\boldsymbol{I}_N}\boldsymbol{\Sigma}^\top\boldsymbol{U}^\top = \frac{1}{N}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top\boldsymbol{U}^\top$$

SVD for PCA

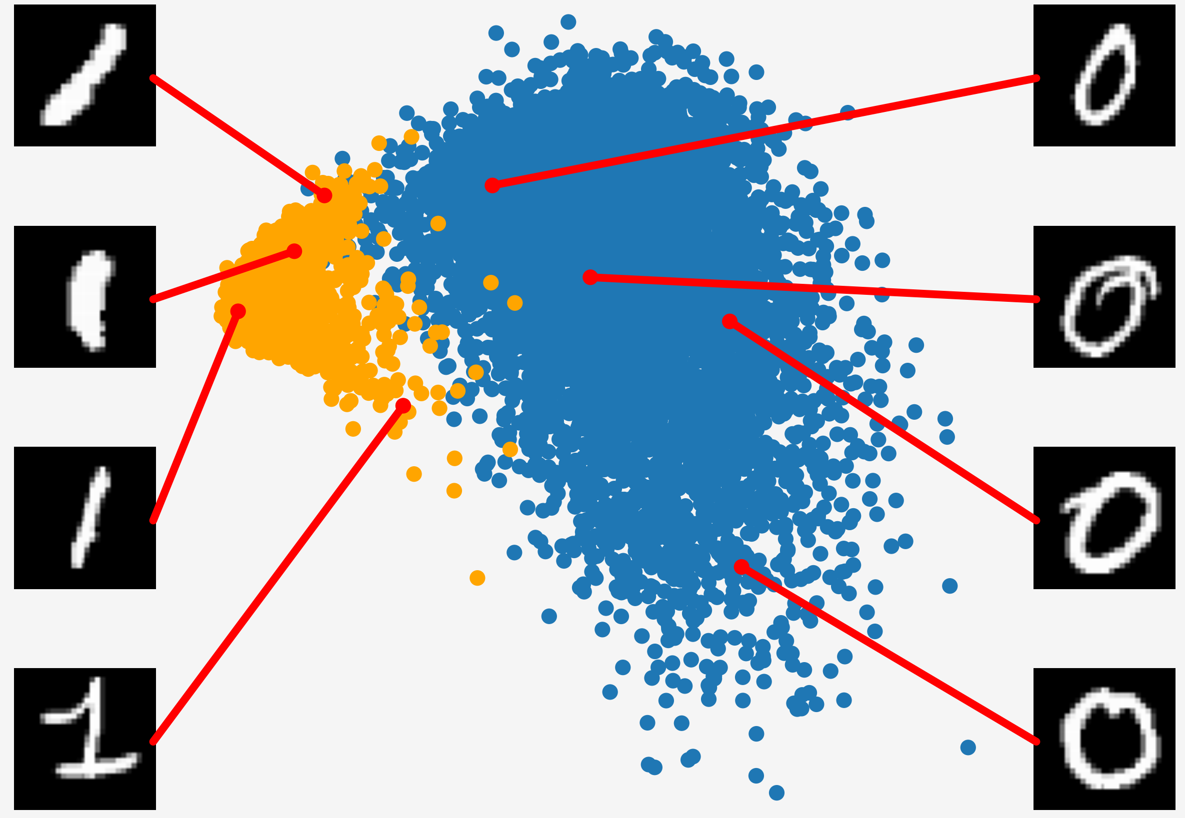
$$\underbrace{\boldsymbol{X}}_{D\times N} = \underbrace{\boldsymbol{U}}_{D\times D}\underbrace{\boldsymbol{\Sigma}}_{D\times N}\underbrace{\boldsymbol{V}^\top}_{N\times N}$$

$$\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top = \frac{1}{N}\boldsymbol{U}\boldsymbol{\Sigma}\underbrace{\boldsymbol{V}^\top\boldsymbol{V}}_{=\boldsymbol{I}_N}\boldsymbol{\Sigma}^\top\boldsymbol{U}^\top = \frac{1}{N}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top\boldsymbol{U}^\top$$

It turns out the columns of ***U*** are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$

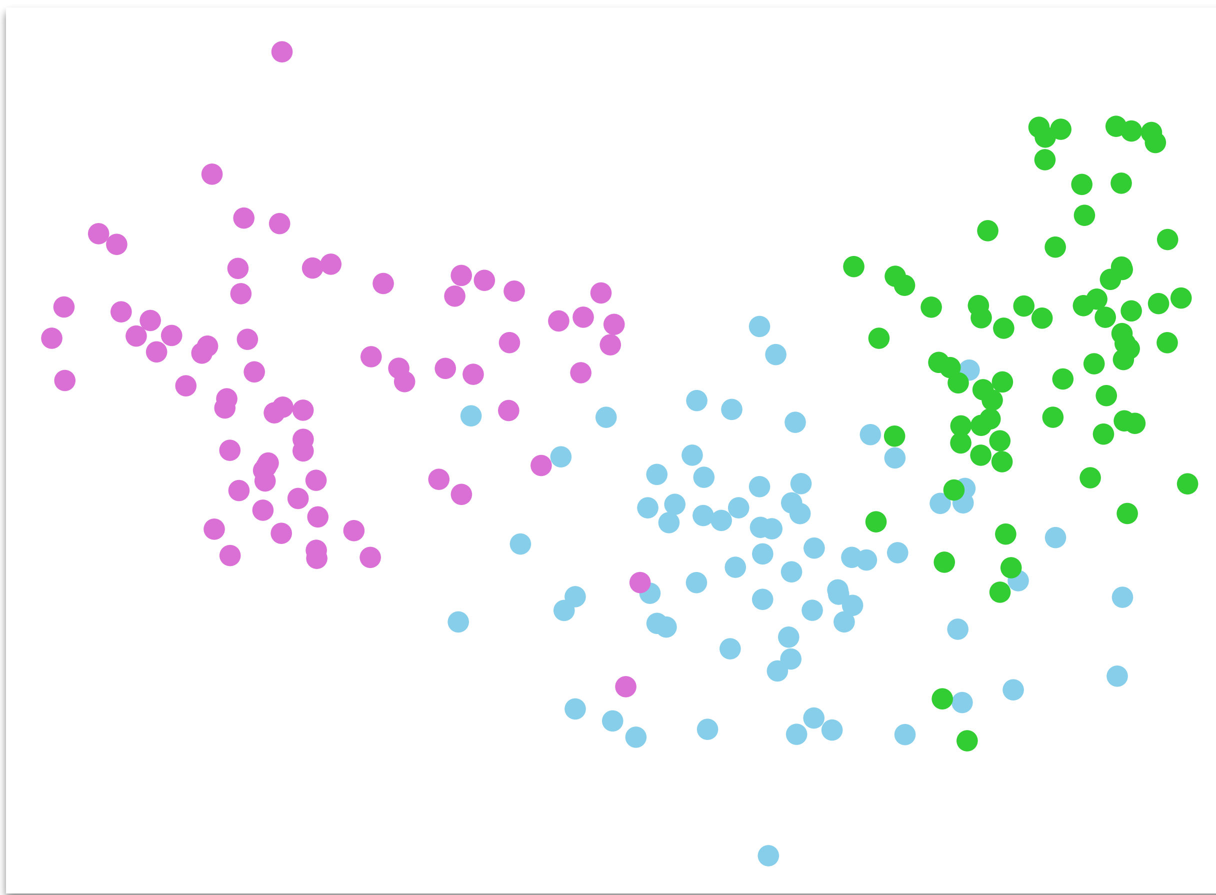
Principal Component Analysis

Example 10.3 (MNIST Digits Embedding)

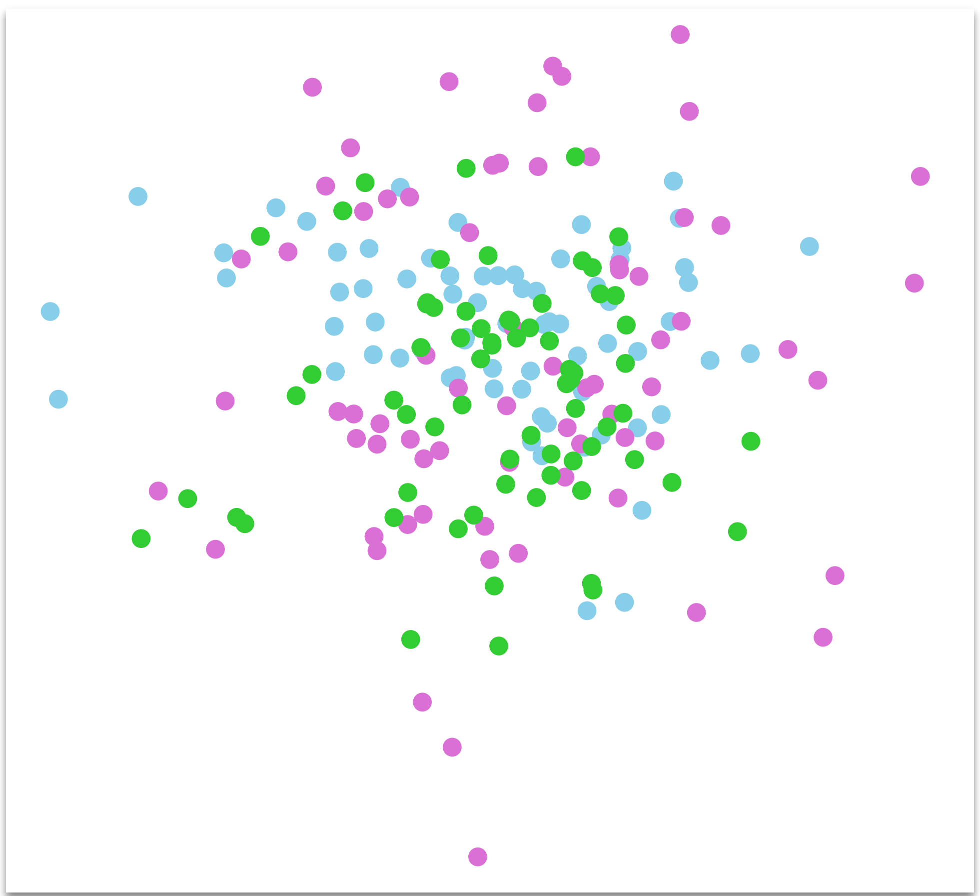


Principal Component Analysis

Top 2 components



Bottom 2 components



Data: three varieties of wheat: Kama, Rosa, Canadian

Attributes: Area, Perimeter, Compactness, Length of Kernel, Width of Kernel, Asymmetry Coefficient, Length of Groove

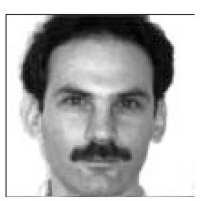
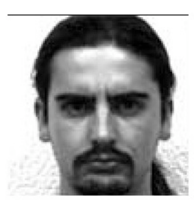
Eigen-faces [Turk & Pentland 1991]

- $d =$ number of pixels
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a face image
- $\mathbf{x}_{ji} =$ intensity of the j-th pixel in image i

Eigen-faces [Turk & Pentland 1991]

- $d =$ number of pixels
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a face image
- $\mathbf{x}_{ji} =$ intensity of the j-th pixel in image i

$$\mathbf{X}_{d\times n} \approxeq \mathbf{U}_{d\times k} \quad \mathbf{Z}_{k\times n}$$

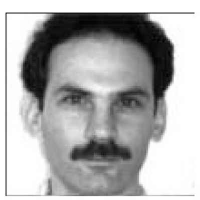
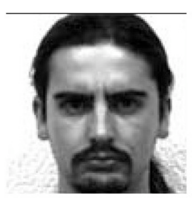
( $\ldots$ ) $\approxeq$ ()

$$\begin{pmatrix} | & & | \\ \mathbf{z}_1 & \ldots & \mathbf{z}_n \\ | & & | \end{pmatrix}$$

Eigen-faces [Turk & Pentland 1991]

- d = number of pixels
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a face image
- $\mathbf{x}_{ji}$ = intensity of the j-th pixel in image i

$$\mathbf{X}_{d\times n} \quad \approx \quad \mathbf{U}_{d\times k} \quad \mathbf{Z}_{k\times n}$$

( $\cdots$ ) $\approx$ () $\begin{pmatrix} | & & | \\ \mathbf{z}_1 & \cdots & \mathbf{z}_n \\ | & & | \end{pmatrix}$

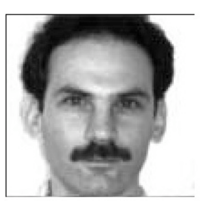
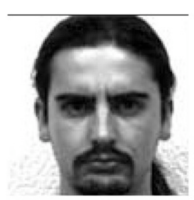
Idea: $\mathbf{z}_i$ more "meaningful" representation of i-th face than $\mathbf{x}_i$

Can use $\mathbf{z}_i$ for nearest-neighbor classification

Eigen-faces [Turk & Pentland 1991]

- $d =$ number of pixels
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a face image
- $\mathbf{x}_{ji} =$ intensity of the j-th pixel in image i

$$\mathbf{X}_{d\times n} \approxeq \mathbf{U}_{d\times k} \quad \mathbf{Z}_{k\times n}$$

( $\dots$ ) $\approxeq$ () $\begin{pmatrix} | & & | \\ \mathbf{z}_1 & \dots & \mathbf{z}_n \\ | & & | \end{pmatrix}$

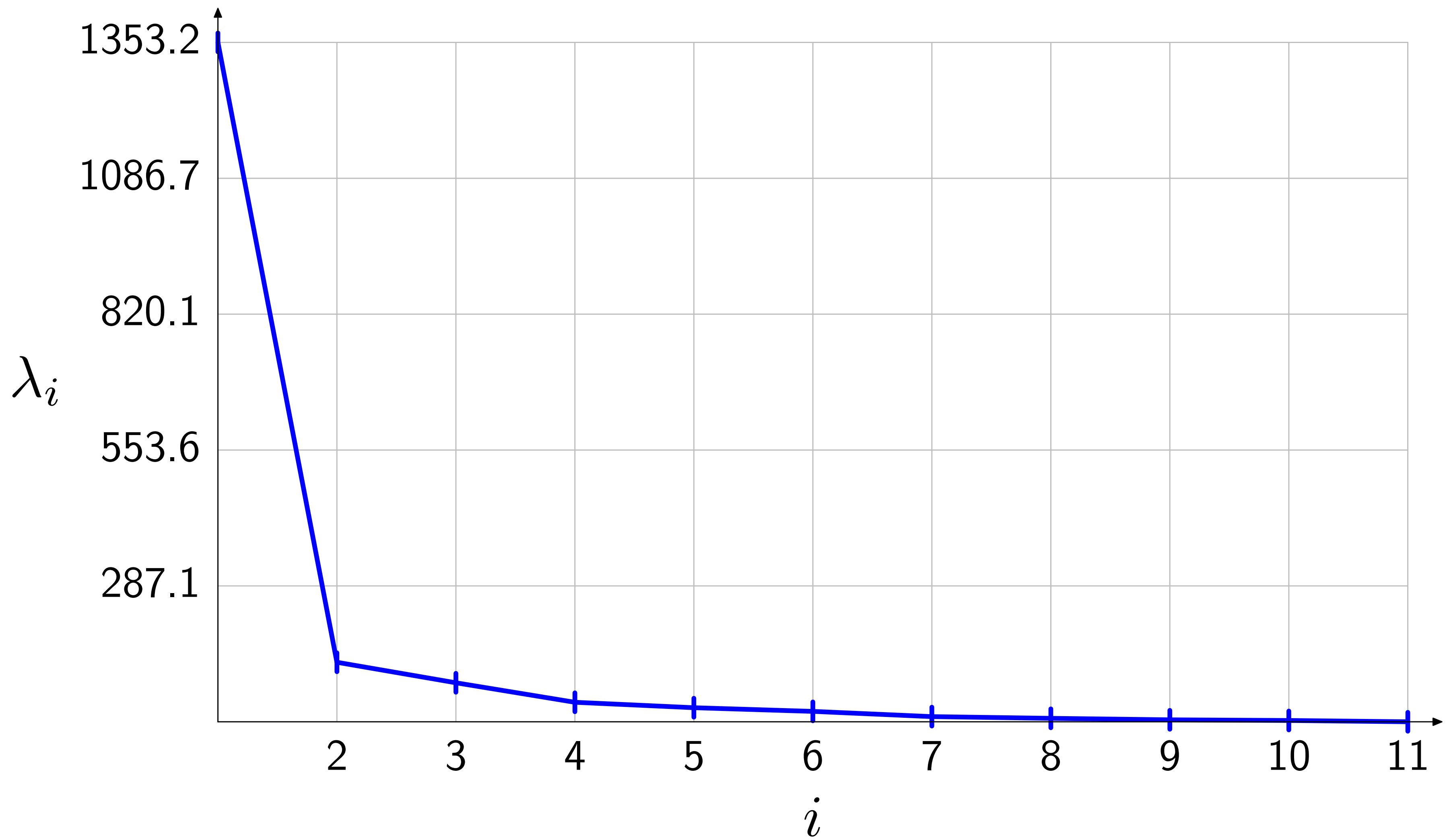
Idea: $\mathbf{z}_i$ more "meaningful" representation of i-th face than $\mathbf{x}_i$

Can use $\mathbf{z}_i$ for nearest-neighbor classification

Much faster: $O(dk + nk)$ time instead of $O(dn)$ when $n, d \gg k$

Aside: How many components?

- Magnitude of eigenvalues indicate fraction of variance captured.
- Eigenvalues on a face image dataset:



- Eigenvalues typically drop off sharply, so don't need that many.
- Of course variance isn't everything...

Latent Semantic Analysis [Deerwater 1990]

- $d =$ number of words in the vocabulary
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of word counts

Latent Semantic Analysis [Deerwater 1990]

- $d =$ number of words in the vocabulary
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of word counts
- $\mathbf{x}_{ji} =$ frequency of word j in document i

$$
\begin{array}{ccccc}
\mathbf{X}_{d\times n} & \approxeq & \mathbf{U}_{d\times k} & & \mathbf{Z}_{k\times n} \\
\left(\begin{array}{rccc}
\text{stocks:} & 2 & \cdots\cdots\cdots & 0 \\
\text{chairman:} & 4 & \cdots\cdots\cdots & 1 \\
\text{the:} & 8 & \cdots\cdots\cdots & 7 \\
\cdots & \vdots & \cdots\cdots\cdots & \vdots \\
\text{wins:} & 0 & \cdots\cdots\cdots & 2 \\
\text{game:} & 1 & \cdots\cdots\cdots & 3
\end{array}\right)
& \approxeq &
\left(\begin{array}{rcr}
0.4 & \cdot\cdot & -0.001 \\
0.8 & \cdot\cdot & 0.03 \\
0.01 & \cdot\cdot & 0.04 \\
\vdots & \cdot\cdot & \vdots \\
0.002 & \cdot\cdot & 2.3 \\
0.003 & \cdot\cdot & 1.9
\end{array}\right)
& &
\left(\begin{array}{ccc}
| & & | \\
\mathbf{z}_1 & \ldots & \mathbf{z}_n \\
| & & |
\end{array}\right)
\end{array}
$$

Latent Semantic Analysis [Deerwater 1990]

- $d =$ number of words in the vocabulary
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of word counts
- $\mathbf{x}_{ji} =$ frequency of word j in document i

$$\mathbf{X}_{d\times n} \quad \approxeq \quad \mathbf{U}_{d\times k} \quad \mathbf{Z}_{k\times n}$$

$$\begin{array}{r} \text{stocks:} \\ \text{chairman:} \\ \text{the:} \\ \cdots \\ \text{wins:} \\ \text{game:} \end{array} \begin{pmatrix} 2 & \cdots\cdots\cdots & 0 \\ 4 & \cdots\cdots\cdots & 1 \\ 8 & \cdots\cdots\cdots & 7 \\ \vdots & \cdots\cdots\cdots & \vdots \\ 0 & \cdots\cdots\cdots & 2 \\ 1 & \cdots\cdots\cdots & 3 \end{pmatrix} \approxeq \begin{pmatrix} 0.4 & \cdot\cdot & -0.001 \\ 0.8 & \cdot\cdot & 0.03 \\ 0.01 & \cdot\cdot & 0.04 \\ \vdots & \cdot\cdot & \vdots \\ 0.002 & \cdot\cdot & 2.3 \\ 0.003 & \cdot\cdot & 1.9 \end{pmatrix} \begin{pmatrix} | & & | \\ \mathbf{z}_1 & \ldots & \mathbf{z}_n \\ | & & | \end{pmatrix}$$

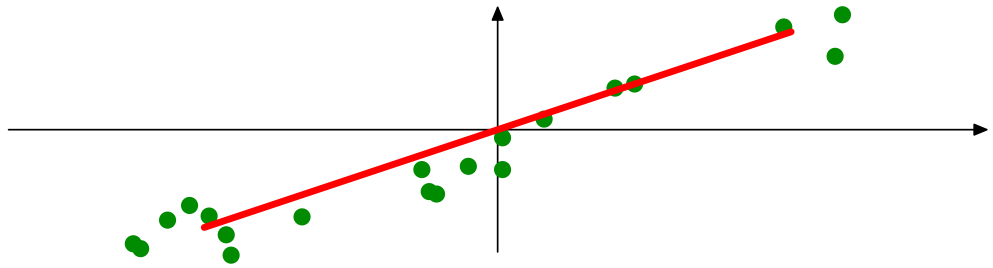
How to measure similarity between two documents?

$\mathbf{z}_1^\top \mathbf{z}_2$ is probably better than $\mathbf{x}_1^\top \mathbf{x}_2$

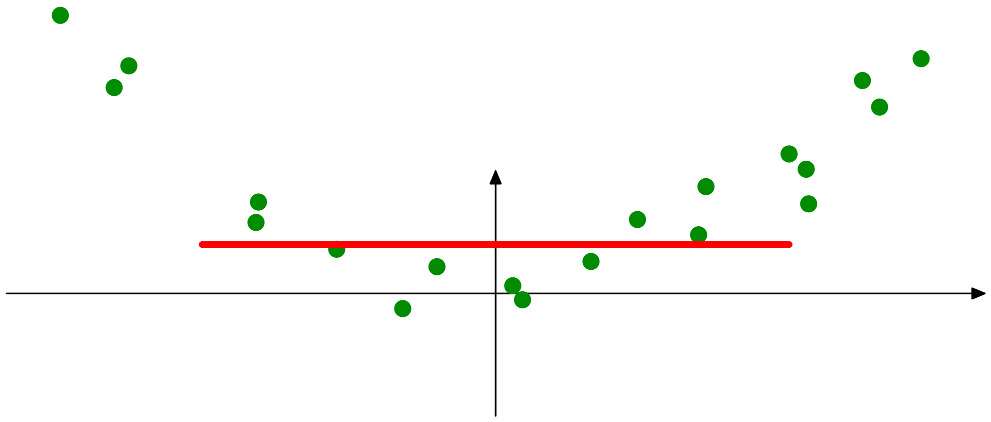
Probabilistic PCA

- If we define a *prior* over *z* then we can **sample** from the latent space and hallucinate images

Limitations of Linearity

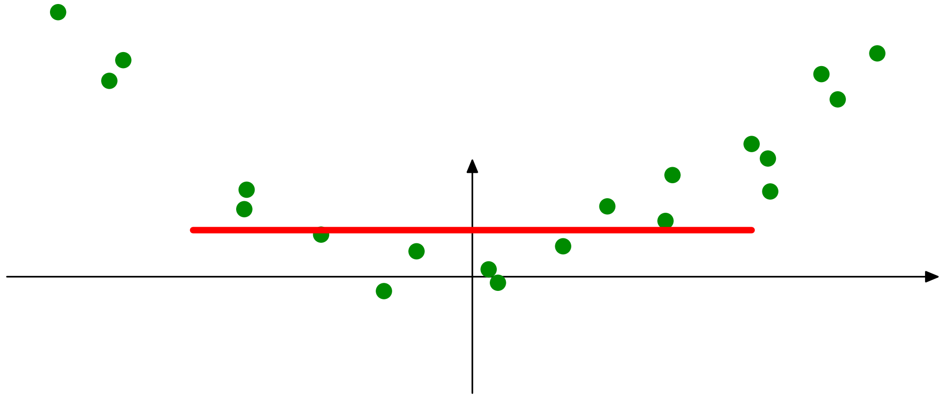


PCA is effective

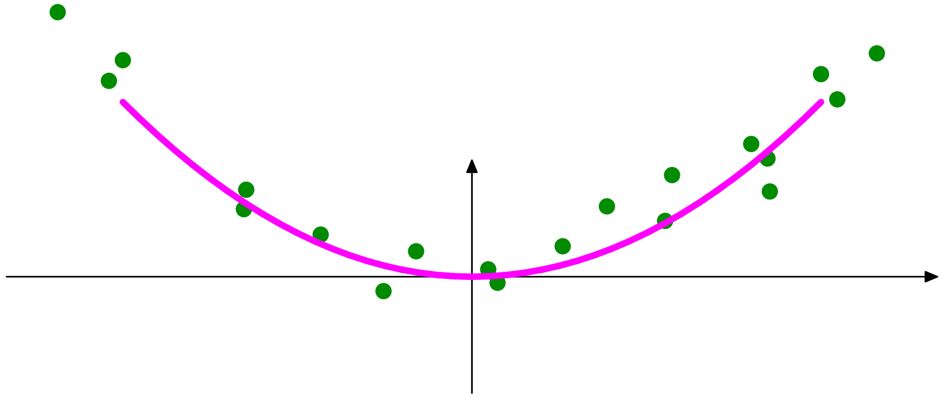


PCA is ineffective

Nonlinear PCA

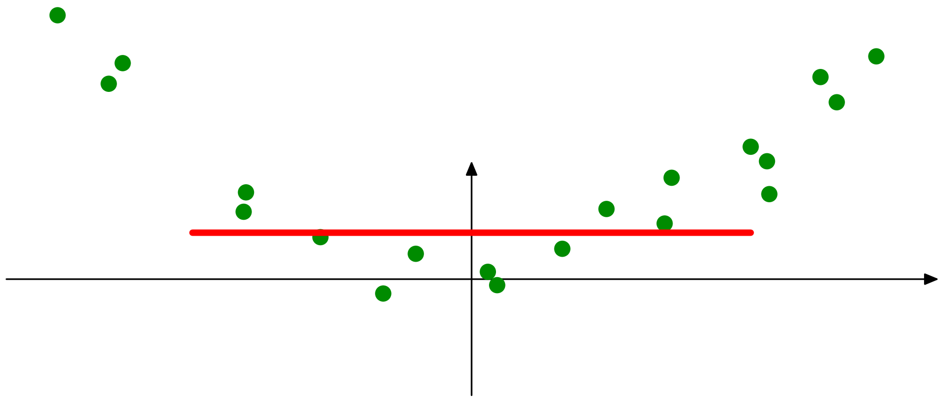


Broken solution

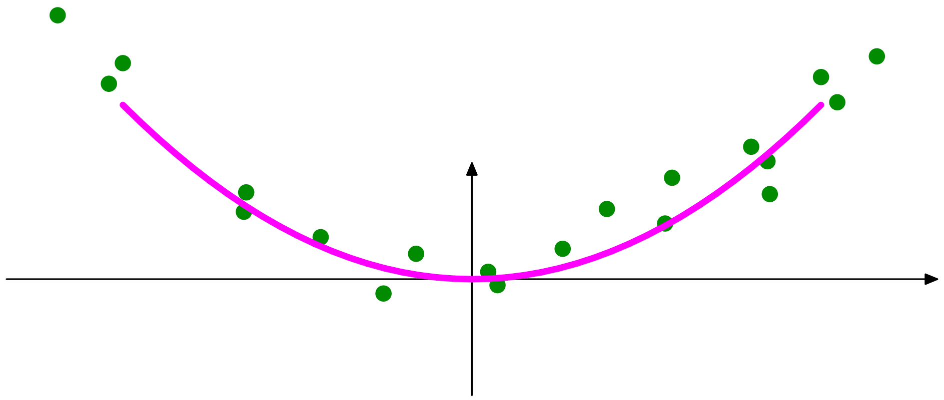


Desired solution

Nonlinear PCA



Broken solution

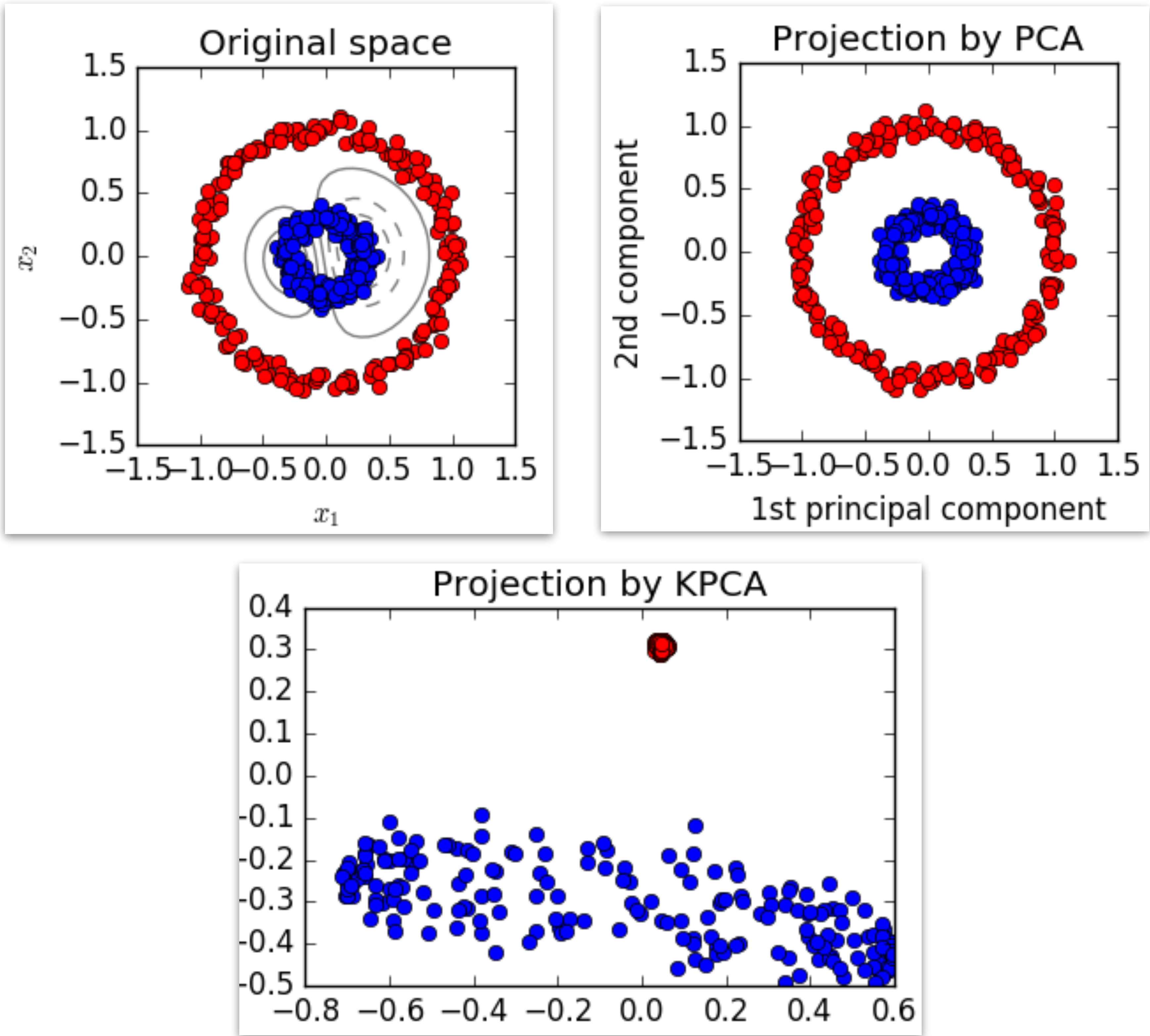


Desired solution

Idea: Use kernels

$$\boxed{\begin{array}{c}\text{Linear dimensionality reduction in } \phi(\mathbf{x}) \text{ space} \\ \Updownarrow \\ \text{Nonlinear dimensionality reduction in } \mathbf{x} \text{ space}\end{array}}$$

Kernel PCA



Original space
x_1
x_2
Projection by PCA
1st principal component
2nd component
Projection by KPCA

Wrapping up

- PCA is a linear model for dimensionality reduction which finds a mapping to a lower dimensional space that maximizes variance

Wrapping up

- PCA is a linear model for dimensionality reduction which finds a mapping to a lower dimensional space that maximizes variance
- We saw that this is equivalent to performing an eigendecomposition on the covariance matrix of $\boldsymbol{X}$

Wrapping up

- PCA is a linear model for dimensionality reduction which finds a mapping to a lower dimensional space that maximizes variance
- We saw that this is equivalent to performing an eigendecomposition on the covariance matrix of $\boldsymbol{X}$
- ***Next time*** Auto-encoders and neural compression for non-linear projections