Dimensionality Reduction

Machine Learning and Pattern Recognition

Prof. Sandra Avila

Institute of Computing (IC/Unicamp)

MC886/MO444, October 3, 2017

Why is Dimensionality Reduction useful?

Why is Dimensionality Reduction useful?

- Data Compression
  - Reduce **time complexity**: less computation required
  - Reduce **space complexity**: less number of features
  - **More interpretable**: it removes noise

Why is Dimensionality Reduction useful?

- Data Compression
  - Reduce **time complexity**: less computation required
  - Reduce **space complexity**: less number of features
  - **More interpretable**: it removes noise
- Data Visualization

Why is Dimensionality Reduction useful?

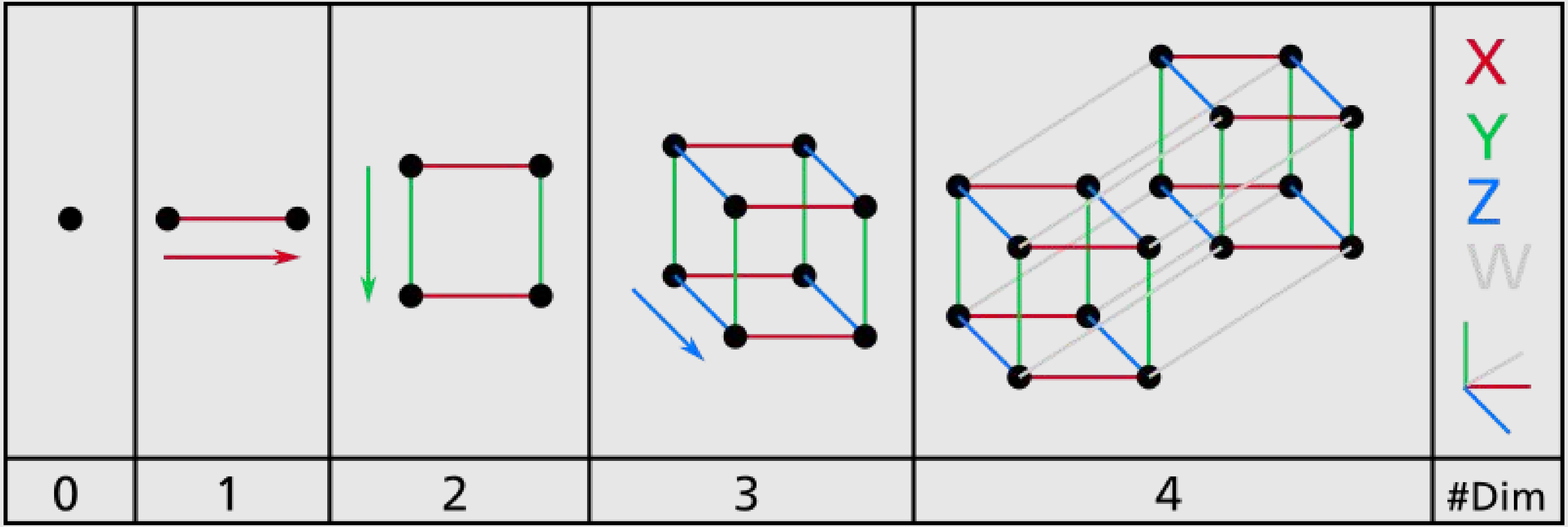
- Data Compression
  - Reduce **time complexity**: less computation required
  - Reduce **space complexity**: less number of features
  - **More interpretable**: it removes noise
- Data Visualization
- To mitigate **“the curse of dimensionality”**

Today’s Agenda

- The Curse of Dimensionality
- PCA (Principal Component Analysis)
  - PCA Formulation
  - PCA Algorithm
  - Choosing k

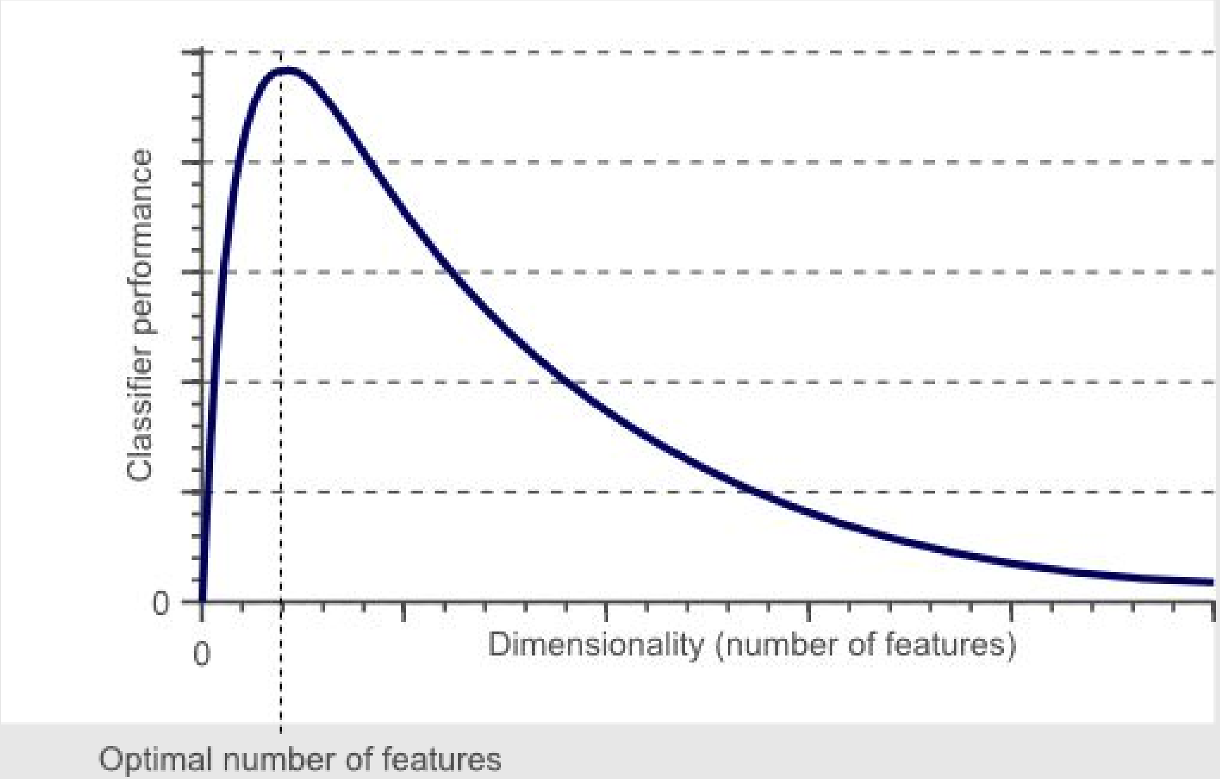
The Curse of Dimensionality

The Curse of Dimensionality



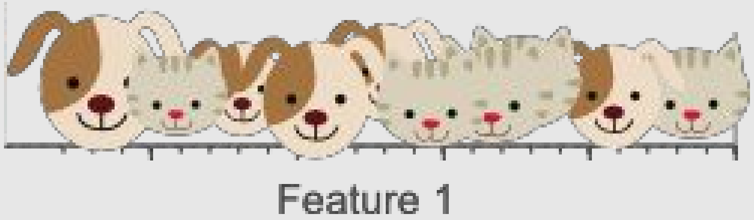
Even a basic 4D hypercube is incredibly hard to picture in our mind.

The Curse of Dimensionality



The Curse of Dimensionality

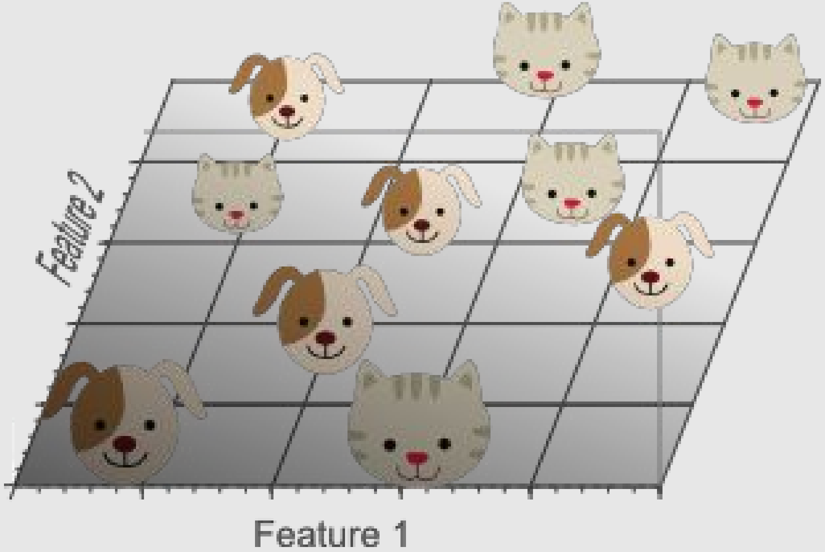
As the dimensionality of data grows, the density of observations becomes lower and lower and lower.



10 images
1 dimension: 5 regions

The Curse of Dimensionality

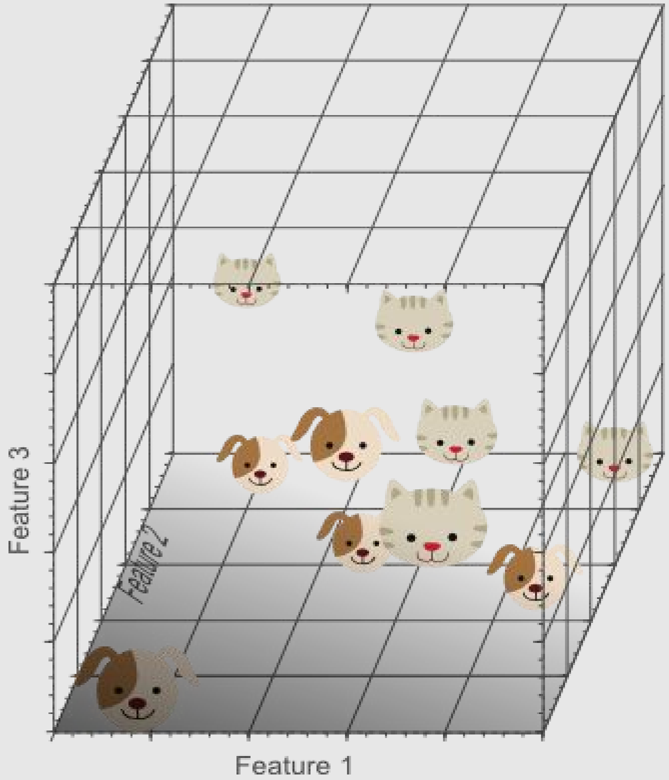
As the dimensionality of data grows, the density of observations becomes lower and lower and lower.



10 images

2 dimensions: 25 regions

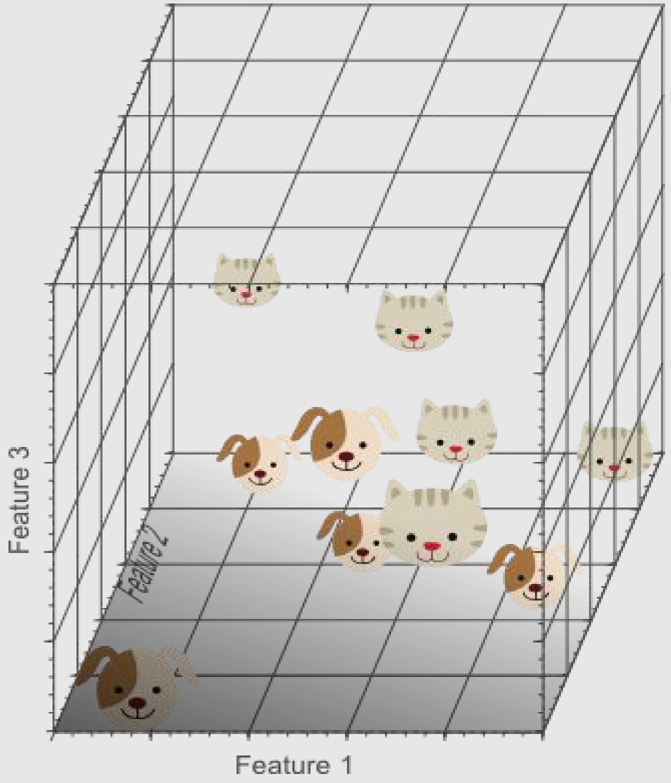
The Curse of Dimensionality



As the dimensionality of data grows, the density of observations becomes lower and lower and lower.

10 images
3 dimensions: 125 regions

The Curse of Dimensionality



- 1 dimension: the sample density is 10/5 = 2 samples/interval
- 2 dimensions: the sample density is 10/25 = 0.4 samples/interval
- 3 dimensions: the sample density is 10/125 = 0.08 samples/interval

The Curse of Dimensionality: Solution?

The Curse of Dimensionality: Solution?

- Increase the size of the training set to reach a sufficient density of training instances.

The Curse of Dimensionality: Solution?

- Increase the size of the training set to reach a sufficient density of training instances.
- Unfortunately, the number of training instances required to reach a given density grows exponentially with the number of dimensions.

How to reduce dimensionality?

How to reduce dimensionality?

- Feature Extraction

- Feature Selection

How to reduce dimensionality?

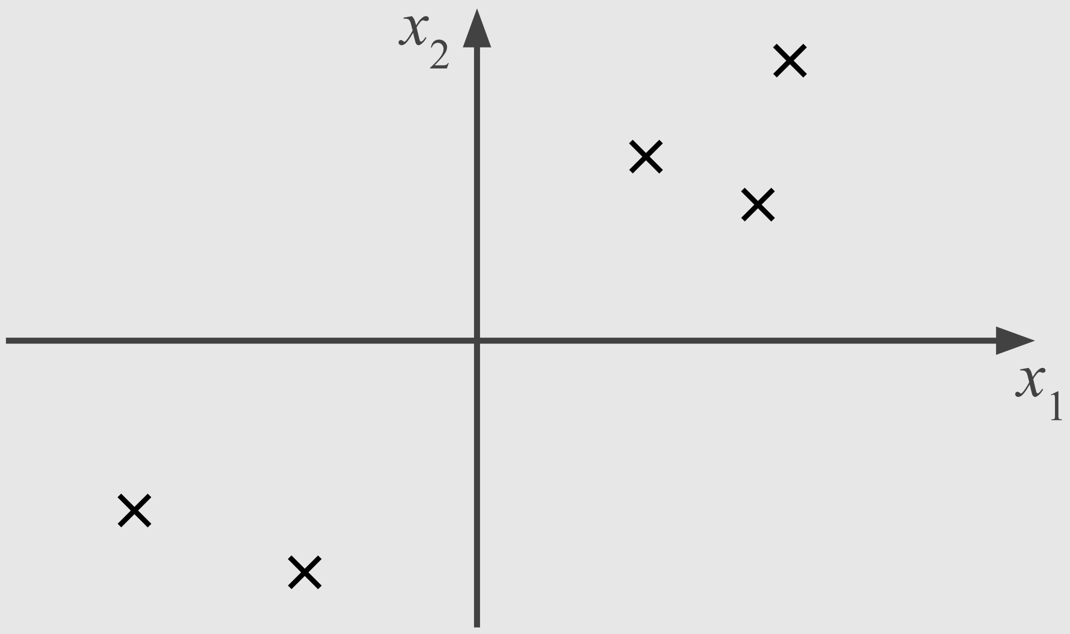
- **Feature Extraction:** create a subset of new features by combining the existing ones.
- **Feature Selection:** choosing a subset of all the features (the ones more informative).

PCA: Principal Component Analysis

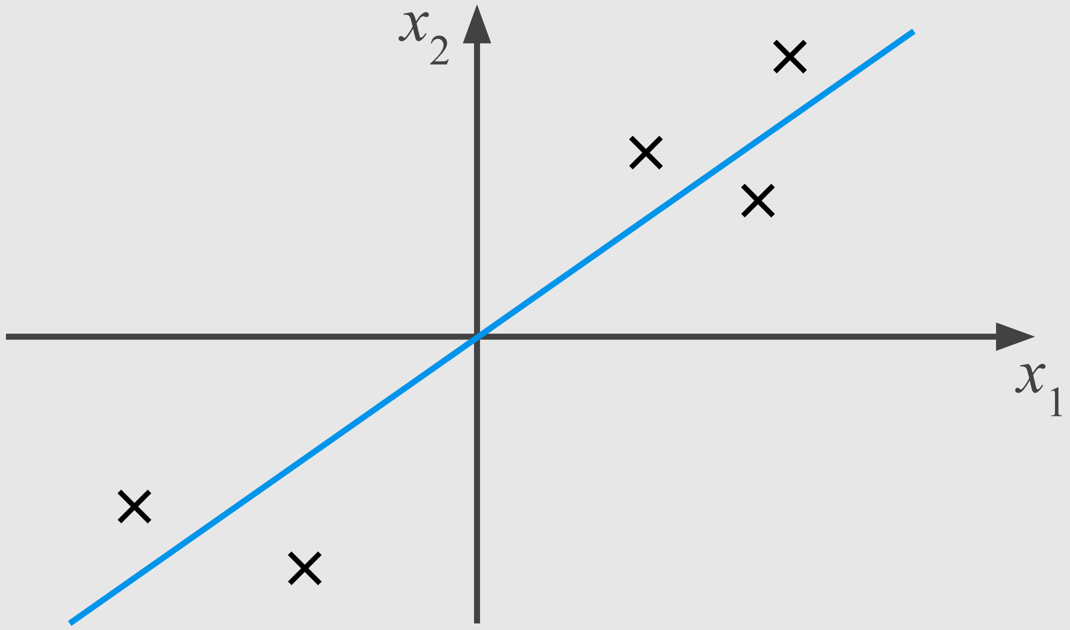
Principal Component Analysis (PCA)

- The most popular dimensionality reduction algorithm.
- PCA have two steps:
  - It identifies the hyperplane that lies closest to the data.
  - It projects the data onto it.

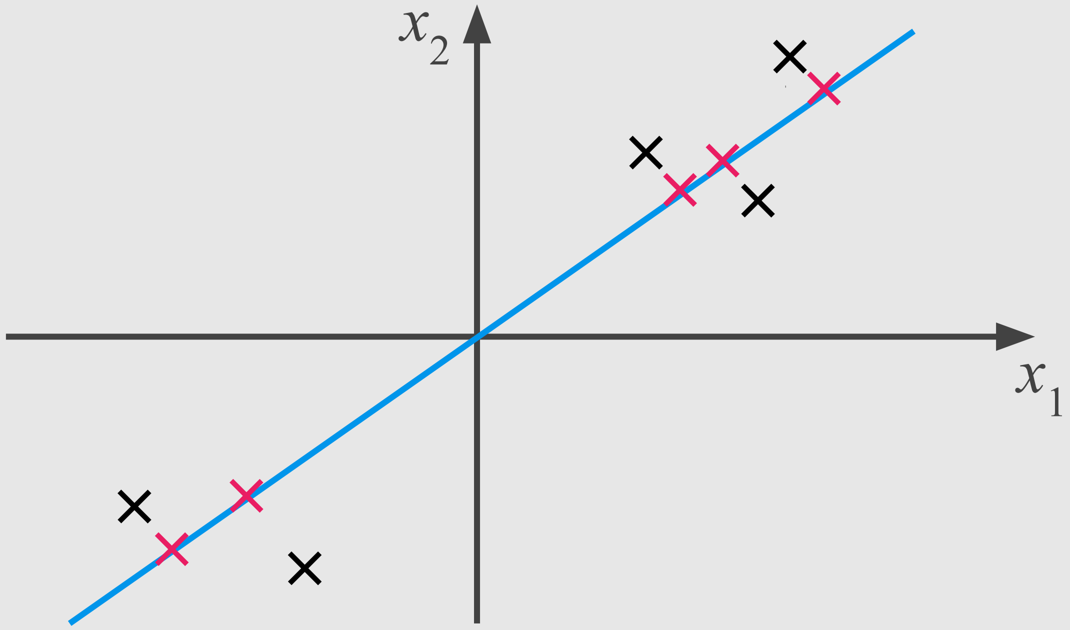
Problem Formulation (PCA)



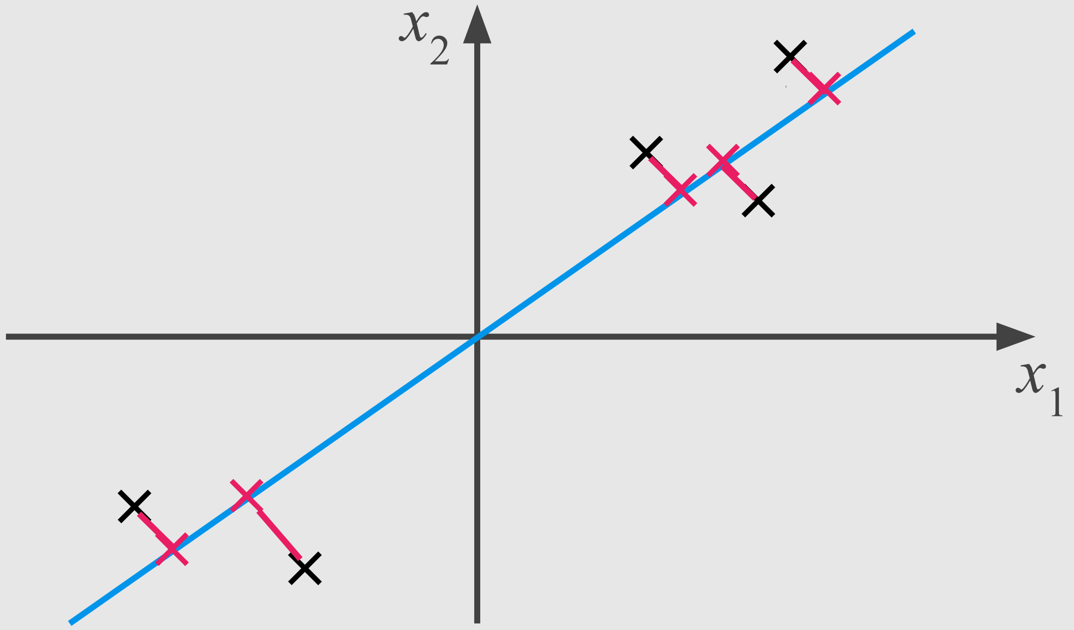
Problem Formulation (PCA)



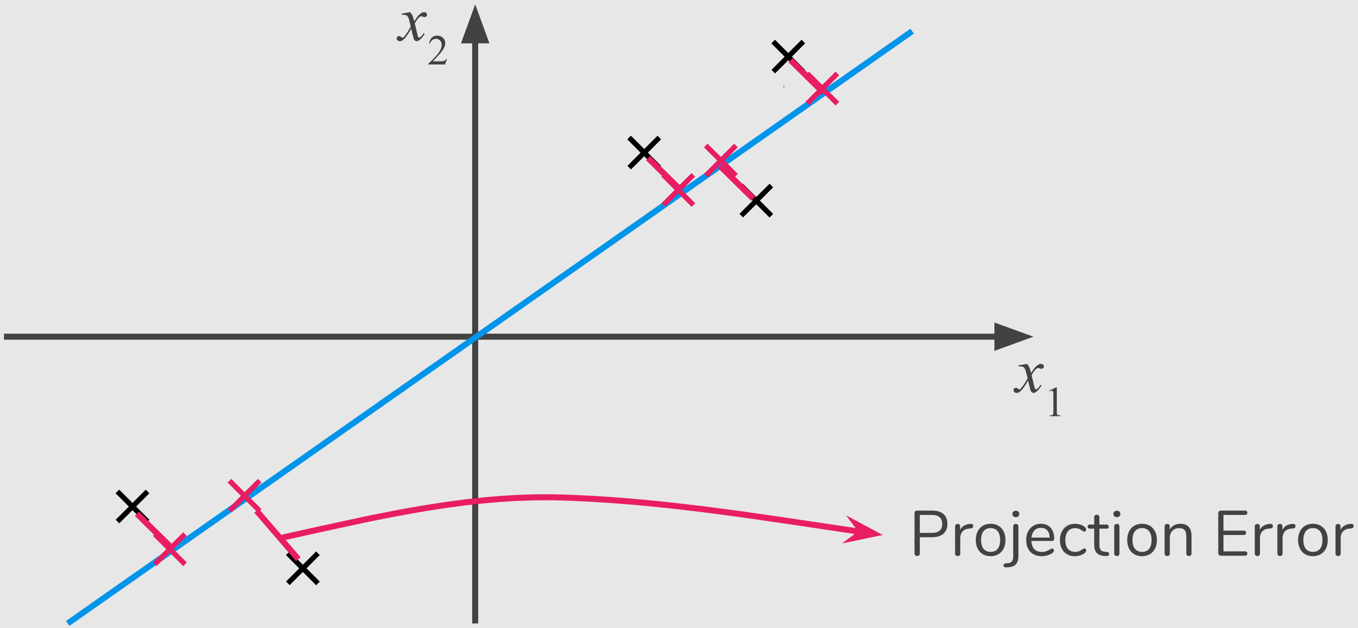
Problem Formulation (PCA)



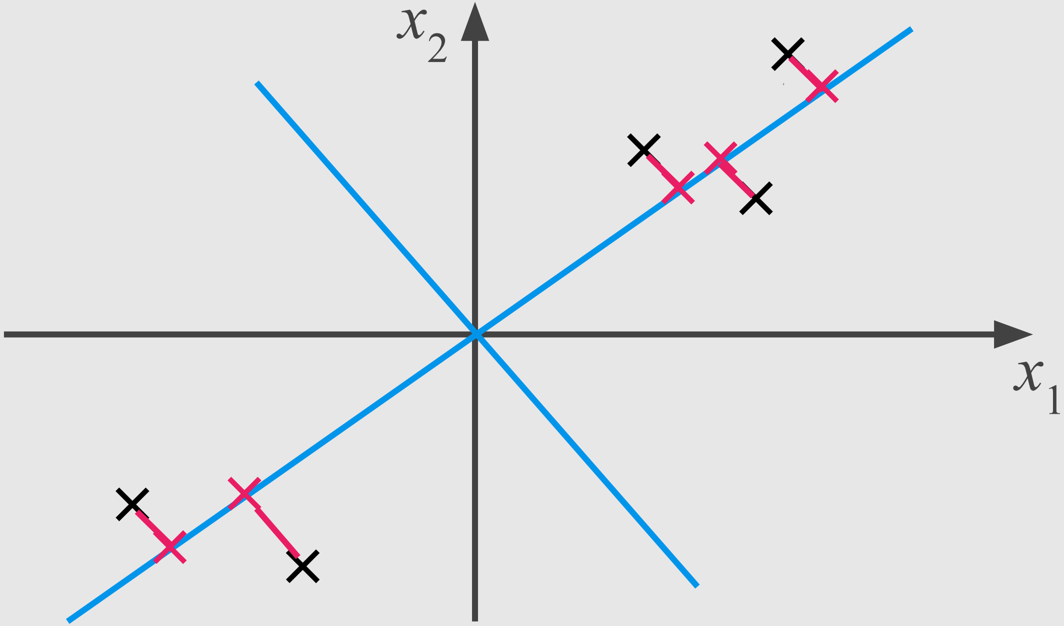
Problem Formulation (PCA)



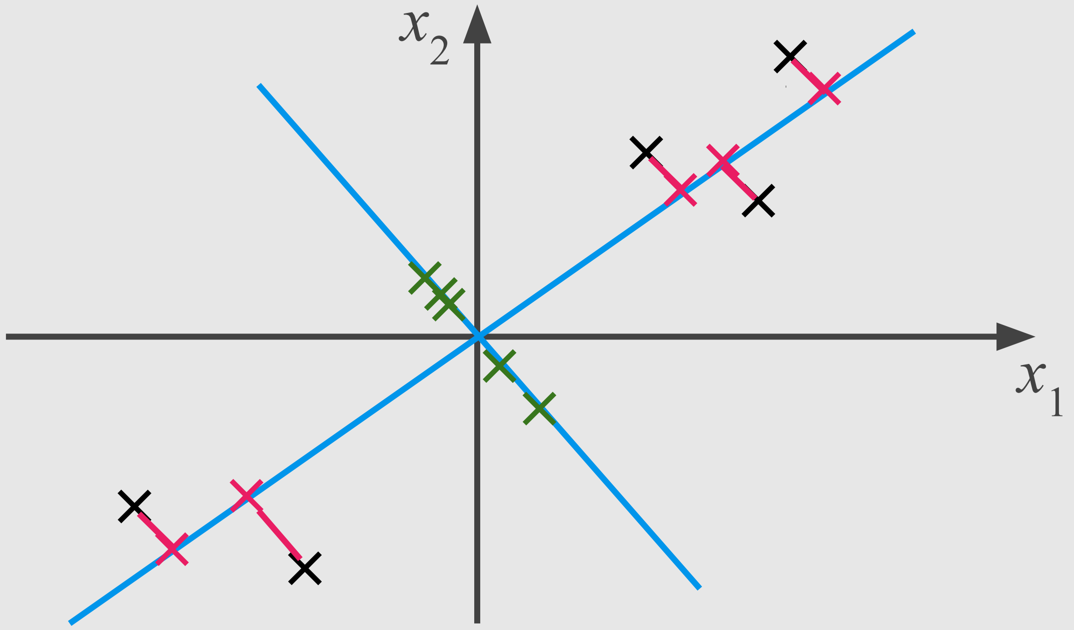
Problem Formulation (PCA)



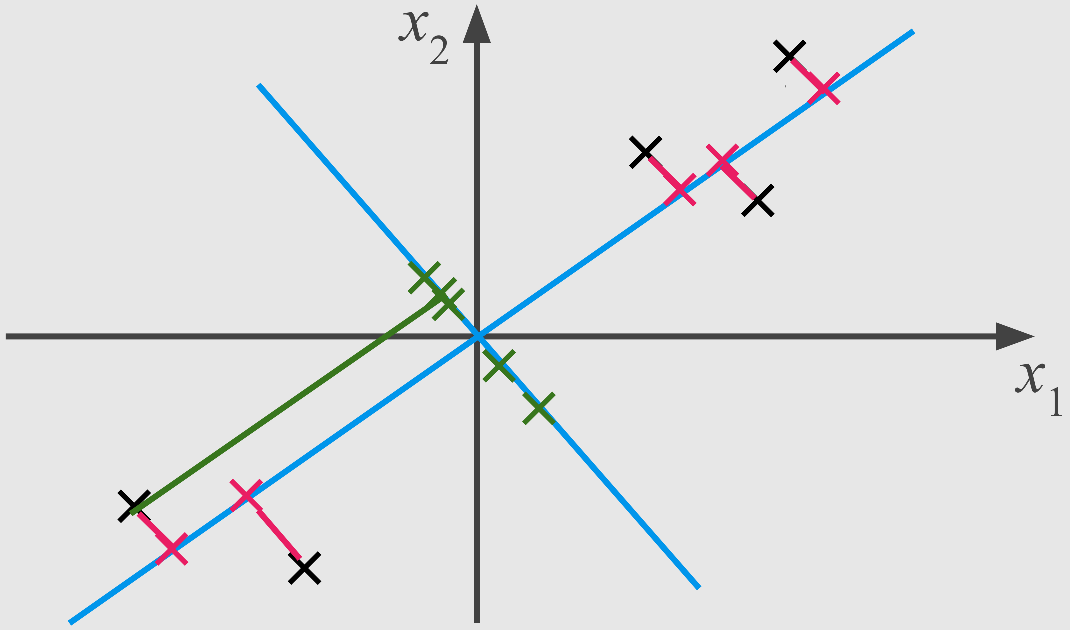
Problem Formulation (PCA)



Problem Formulation (PCA)

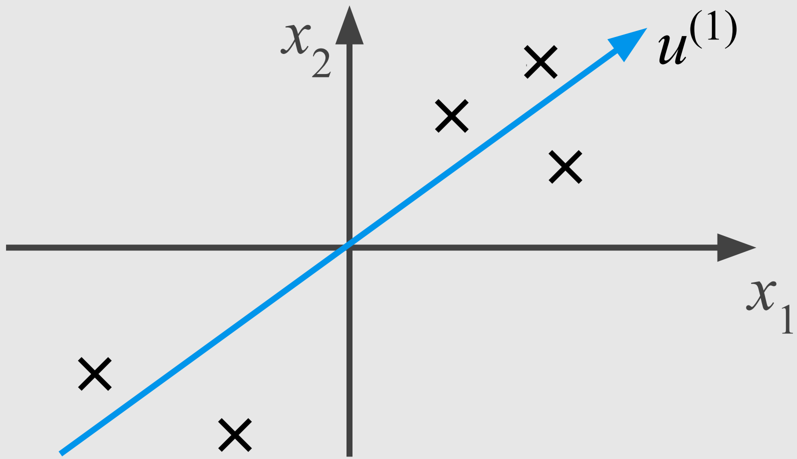


Problem Formulation (PCA)



Problem Formulation (PCA)

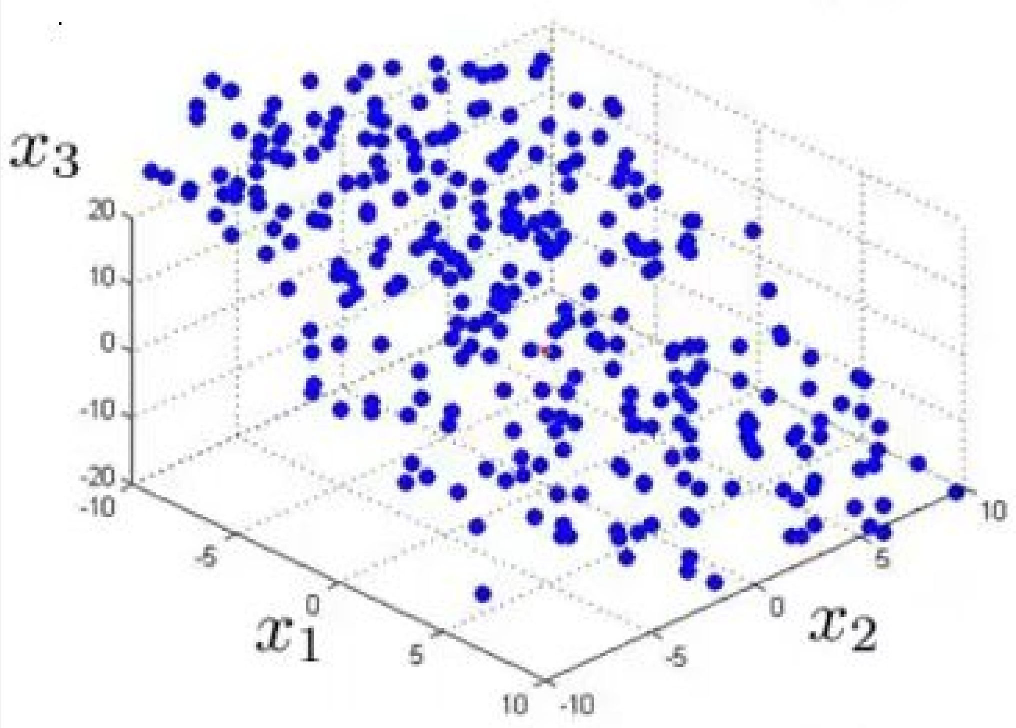
- Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.



Problem Formulation (PCA)

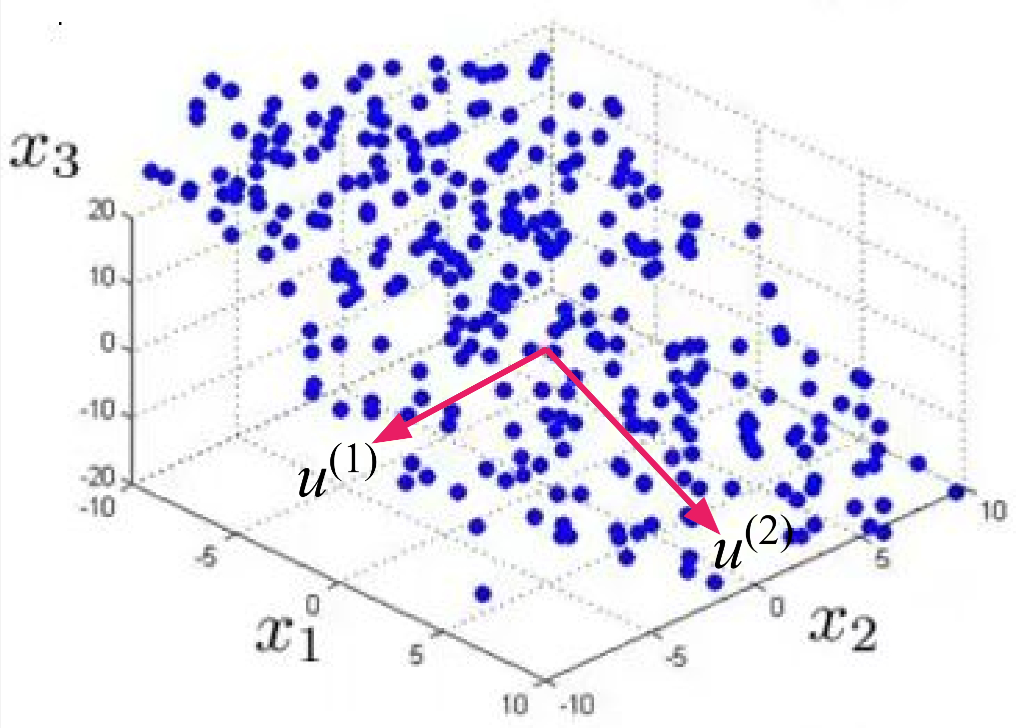
- Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Problem Formulation (PCA)

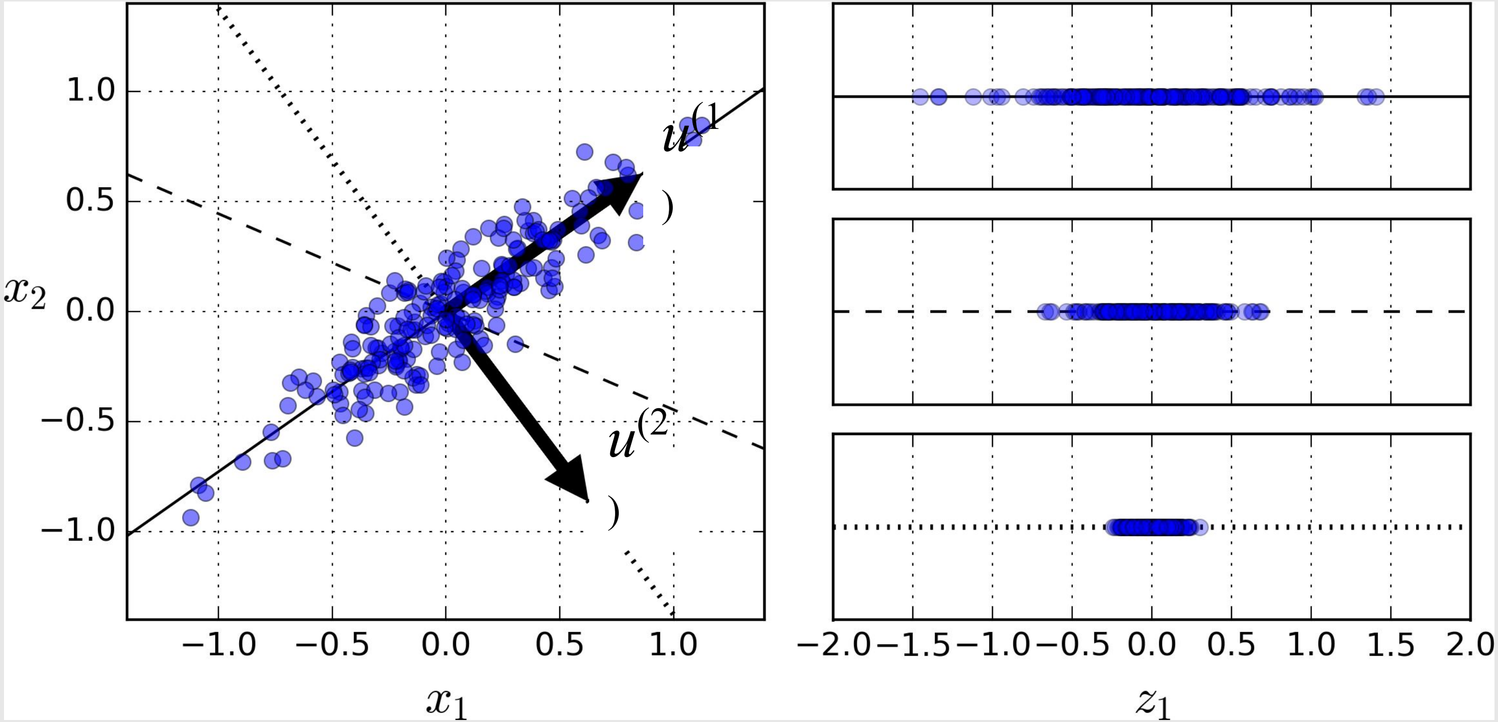


3d ⇨ 2d

Problem Formulation (PCA)



Preserving the Variance



PCA Algorithm

Data Preprocessing

Training set: $x^{(1)}, x^{(2)}, ..., x^{(m)}$

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

Data Preprocessing

Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

If different features on different scales, scale features to have comparable range of values.

PCA Algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^{\mathrm{T}}$$

PCA Algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^{\mathrm{T}} \Rightarrow n \times n \text{ matrix}$$

PCA Algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^{\mathrm{T}} \Rightarrow n \times n \text{ matrix}$$

Compute "eigenvectors" of matrix $\boldsymbol{\Sigma}$:

[U, S, V] = svd(sigma) $\Rightarrow$ Singular Value Decomposition

PCA Algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^{\mathrm{T}} \quad \Rightarrow \quad n \times n \text{ matrix}$$

Compute "eigenvectors" of matrix $\mathbf{\Sigma}$:

$$[\mathbf{U}, S, V] = \text{svd(sigma)} \quad \Rightarrow \quad \text{Singular Value Decomposition}$$

PCA Algorithm

From $[\mathrm{U}, \mathrm{S}, \mathrm{V}]$ = svd(sigma), we get:

$$U = \begin{bmatrix} | & & | \\ u^{(1)} & \cdots & u^{(n)} \\ | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

PCA Algorithm

From [U, S, V] = svd(sigma), we get:

$$U = \begin{bmatrix} | & & | \\ \underbrace{u^{(1)} \cdots}_{k} & & u^{(n)} \\ | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

PCA Algorithm

From [U, S, V] = svd(sigma), we get:

$$U = \begin{bmatrix} | & | & | \\ u^{(1)} & \cdots & u^{(n)} \\ | & | & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

$$\underbrace{u^{(1)} \cdots}_{k}$$

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

$$z = \underbrace{\begin{bmatrix} | & | & | \\ u^{(1)} & \cdots & u^{(k)} \\ | & | & | \end{bmatrix}^{\mathrm{T}}}_{k \times n} \underbrace{x}_{n \times 1}$$

PCA Algorithm

After mean normalization and optionally feature scaling:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^{\mathrm{T}}$$

$[\mathrm{U}, \mathrm{S}, \mathrm{V}] =$ svd(sigma)

$$z = (\mathrm{U}_{\mathrm{reduce}})^{\mathrm{T}} \times x$$

Choosing the Number of Principal Components

Choosing k (#Principal Components)

Typically, choose k to be smallest value so that:

$$\frac{1/m \sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{1/m \sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01$$

“99% of variance is retained”

Choosing k (#Principal Components)

Typically, choose k to be smallest value so that:

$$\frac{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01$$

→ Average squared projection error

→ Total variation in the data

"99% of variance is retained"

Choosing k (#Principal Components)

[U, S, V] = svd(sigma)

$$\frac{1/m \sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{1/m \sum_{i=1}^{m} \|x^{(i)}\|^2} \Rightarrow 1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}}$$

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 8 “Dimensionality Reduction”
- Pattern Recognition and Machine Learning, Chap. 12 “Continuous Latent Variables”
- Pattern Classification, Chap. 10 “Unsupervised Learning and Clustering”

Machine Learning Courses

- https://www.coursera.org/learn/machine-learning, Week 8