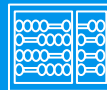




recod.ai  
reasoning for complex data



# Dimensionality Reduction (PCA)

## Machine Learning

**Prof. Sandra Avila**

Institute of Computing (IC/Unicamp)

MC886/MO444, October 13, 2022

# 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**”

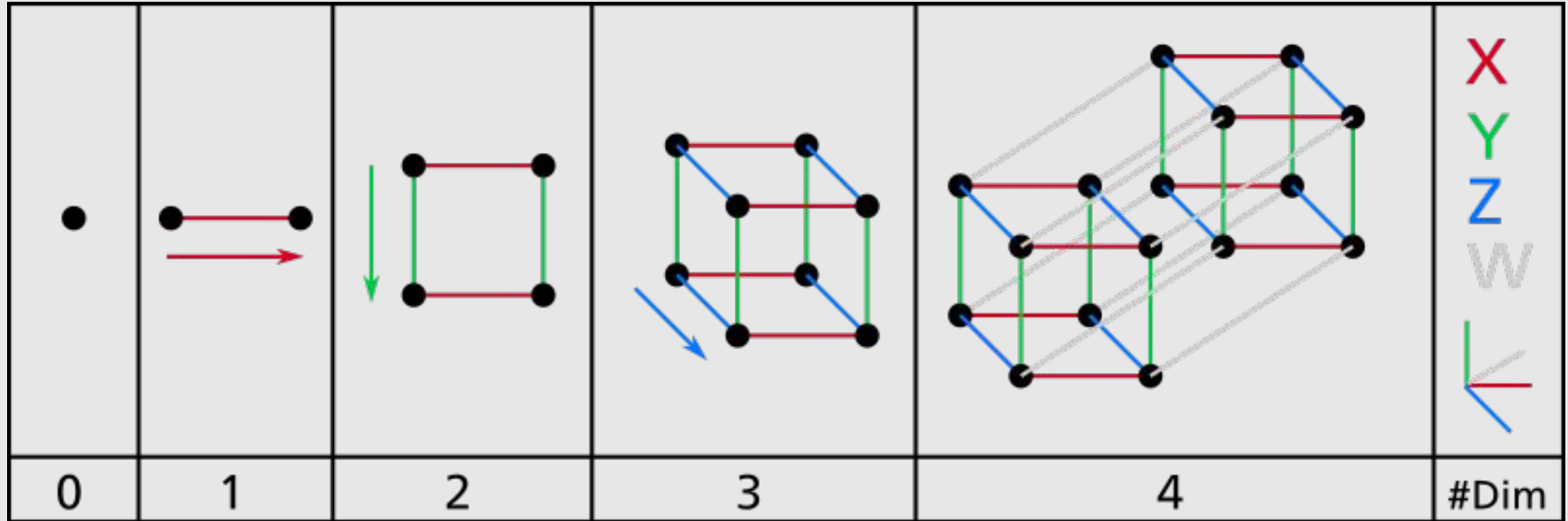
# Today's Agenda

— — —

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

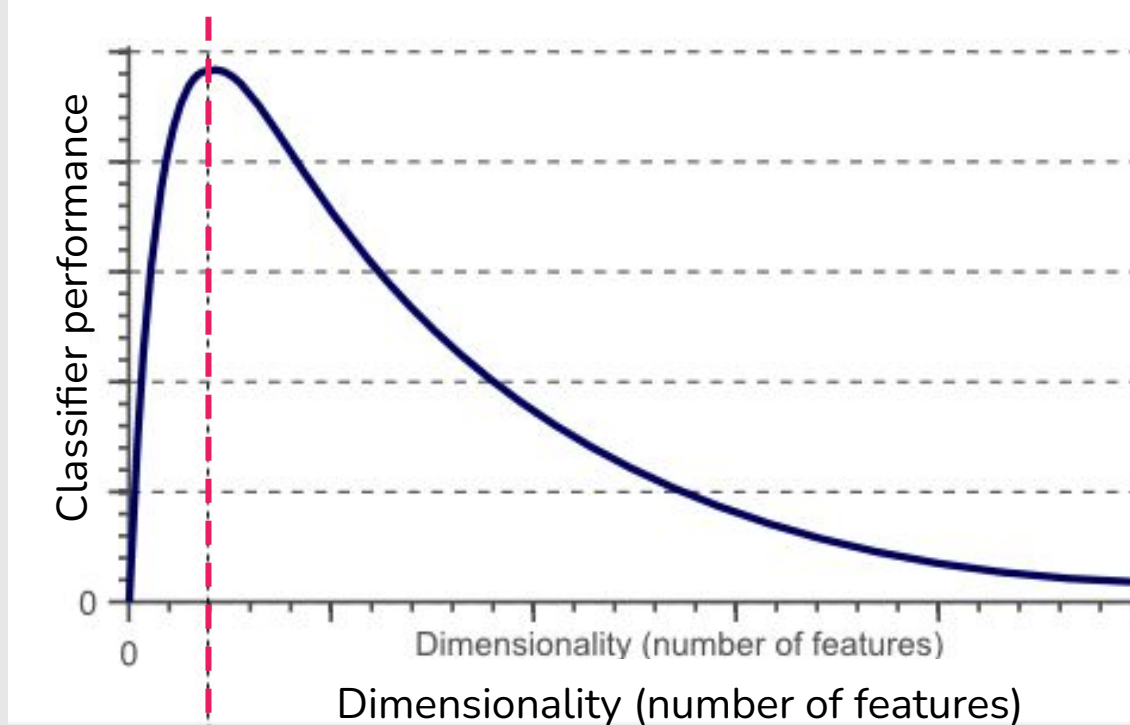
# The Curse of Dimensionality

# The Curse of Dimensionality



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

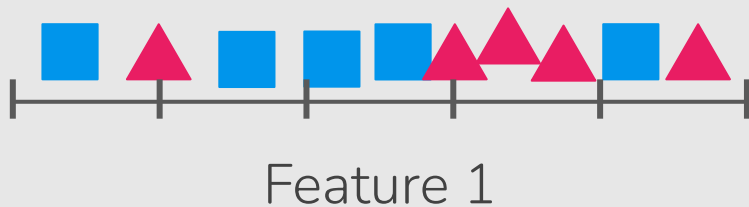
# The Curse of Dimensionality



Optimal number of features

# The Curse of Dimensionality

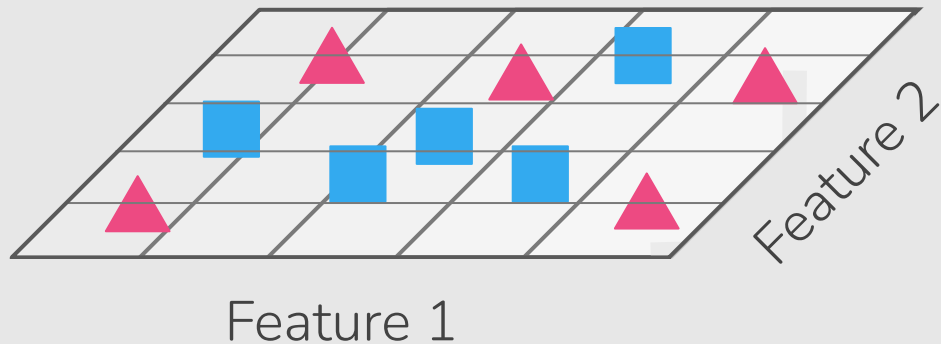
As the dimensionality of data grows, the density of observations becomes lower and lower and lower.



10 samples  
1 dimension: 5 regions

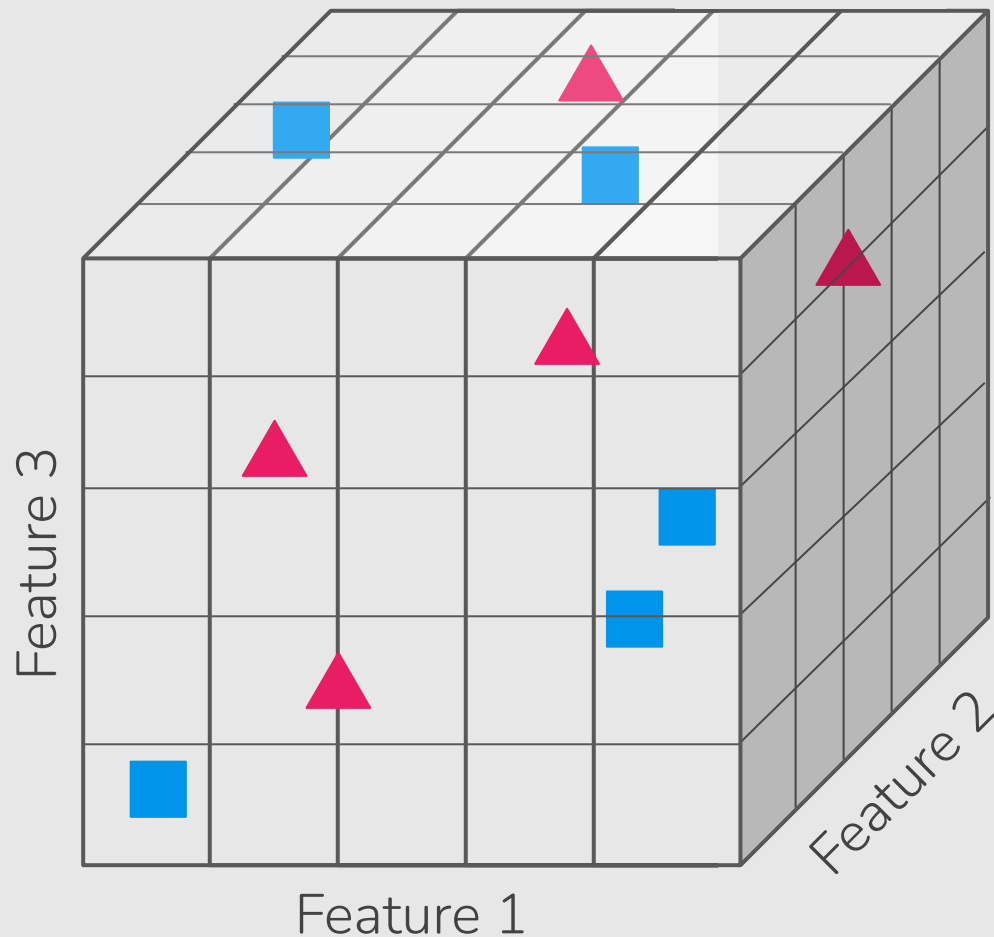
# The Curse of Dimensionality

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



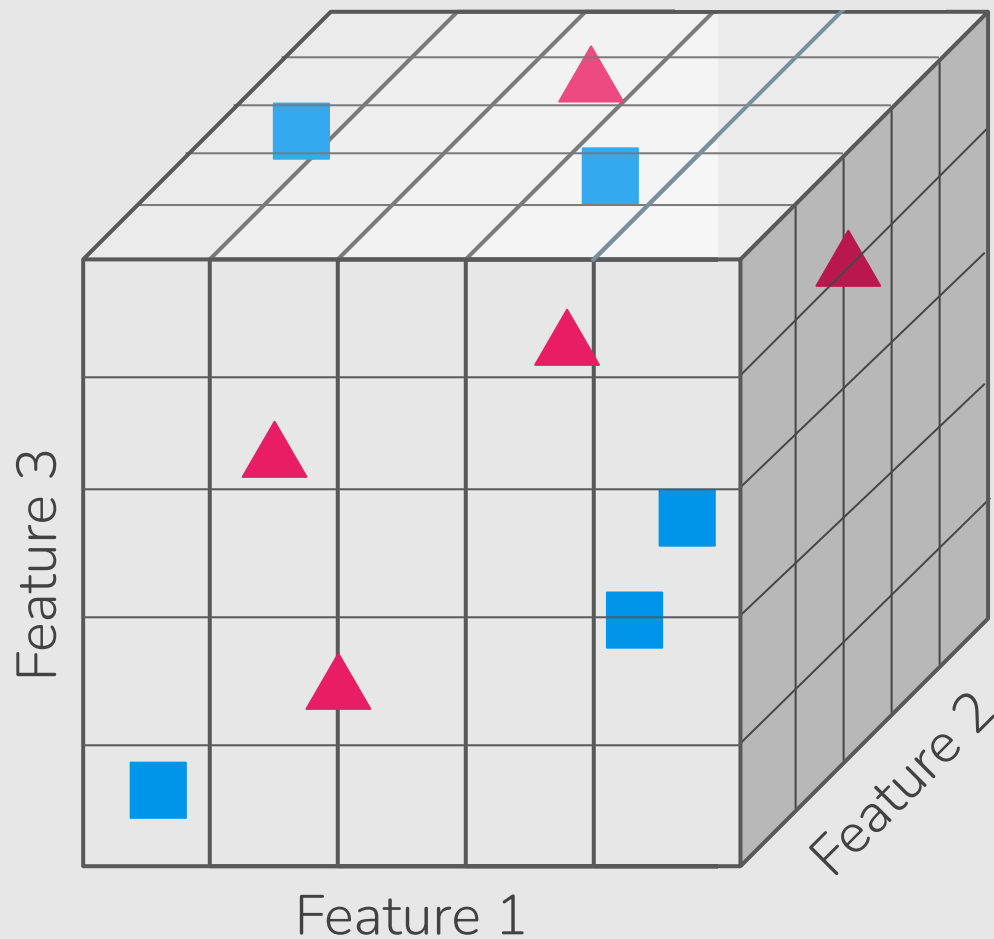
10 samples  
2 dimensions: 25 regions





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

10 samples  
3 dimensions: 125 regions



- 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?

- **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?

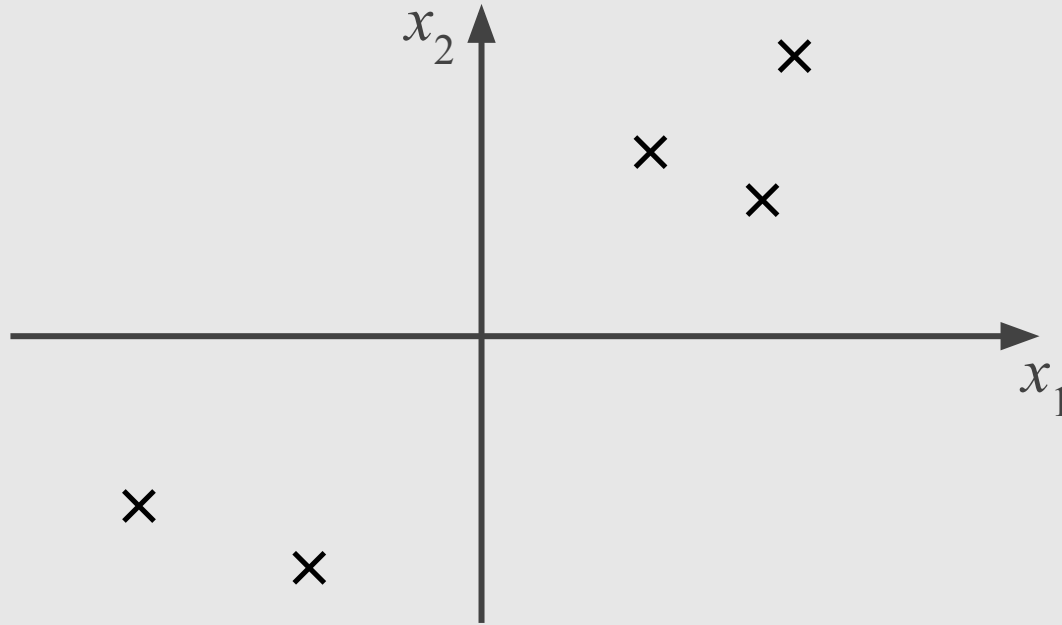
- **Feature Selection:** choosing a subset of all the features (the ones more informative).
  - $\mathbf{x}_1, x_2, \mathbf{x}_3, x_4, \mathbf{x}_5$
- **Feature Extraction:** create a subset of new features by combining the existing ones.
  - $z = f(x_1, x_2, x_3, x_4, x_5)$

# PCA: Principal Component Analysis

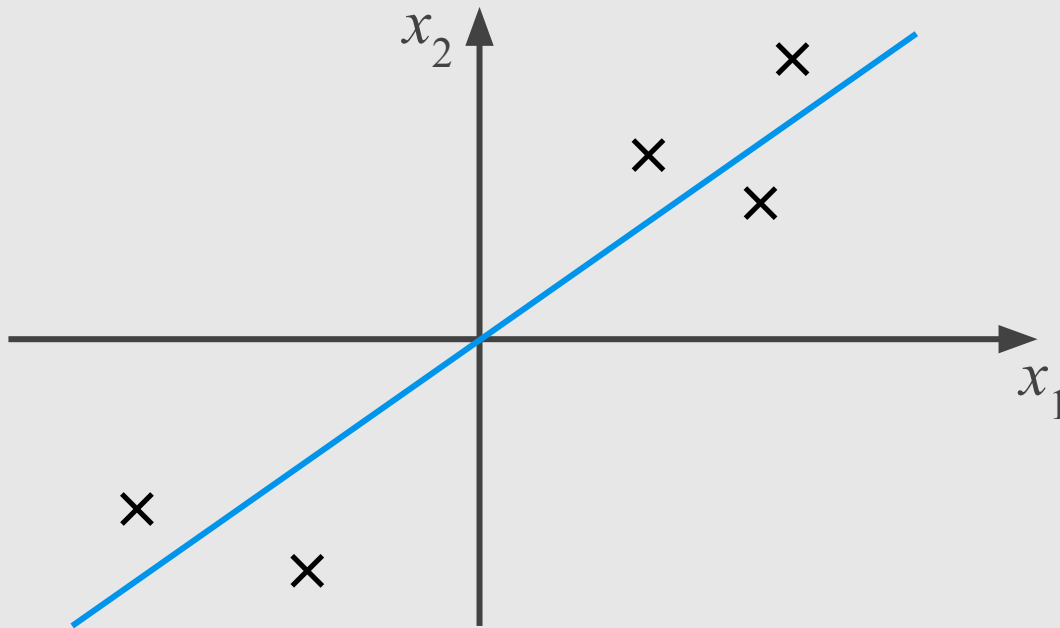
# Principal Component Analysis (PCA)

- 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)

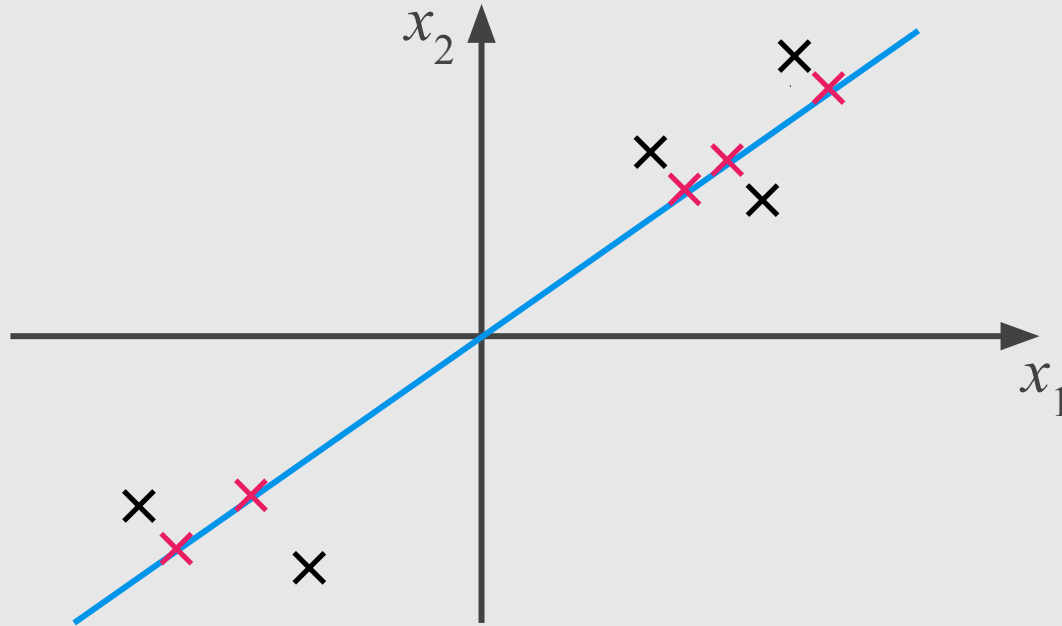


# Problem Formulation (PCA)

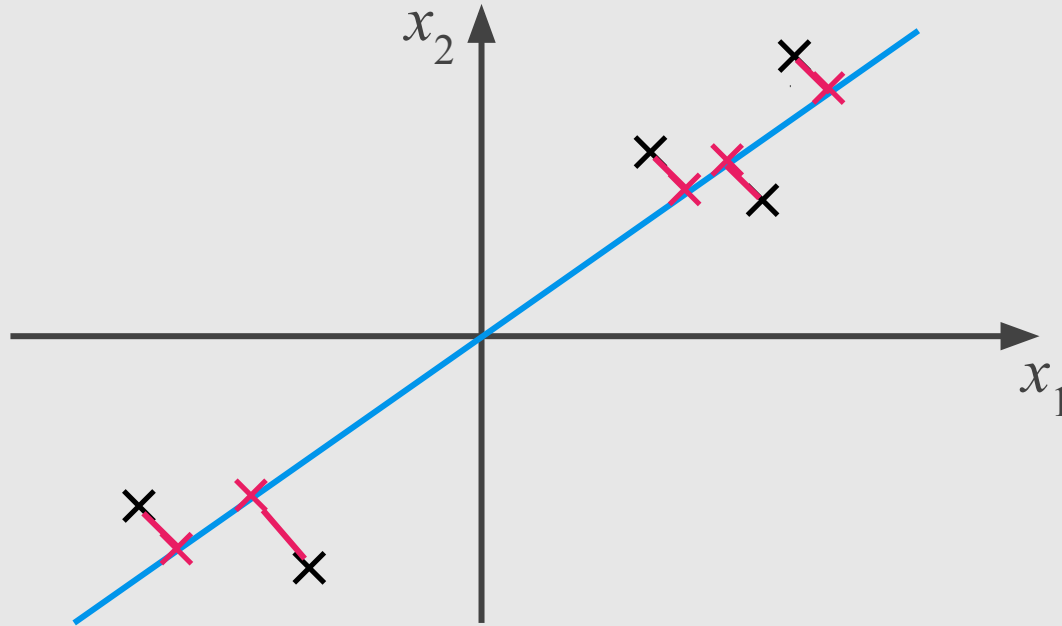




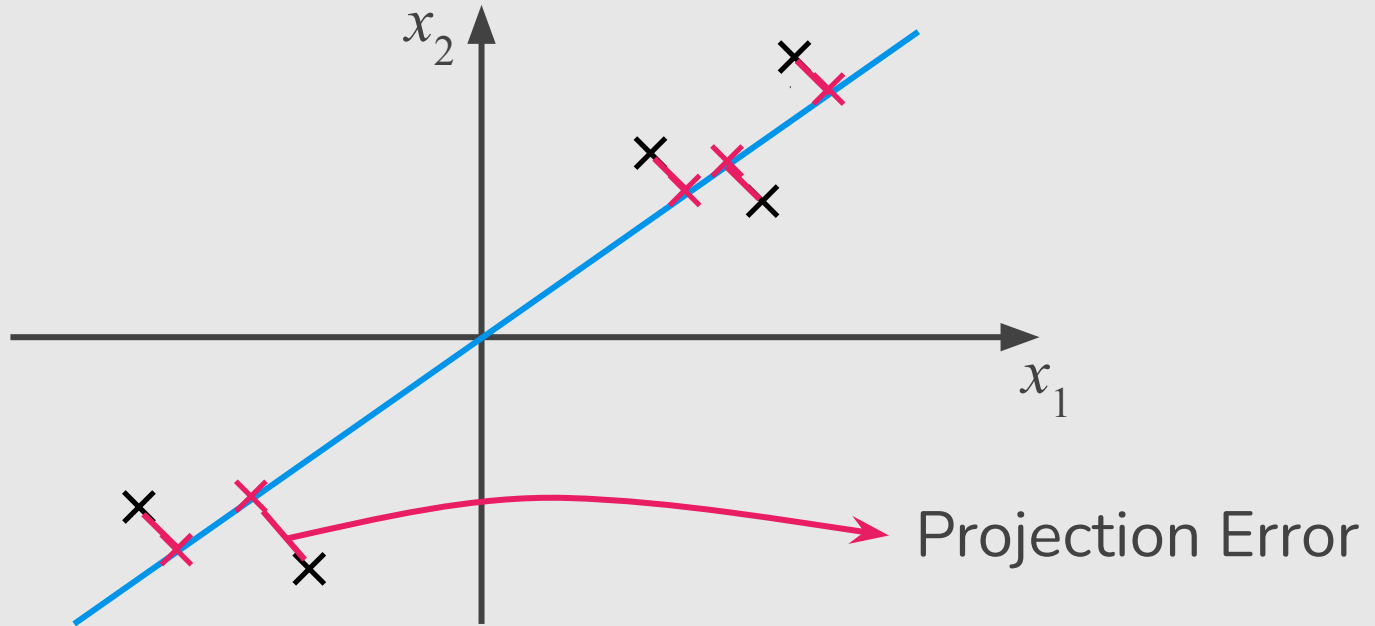
# Problem Formulation (PCA)



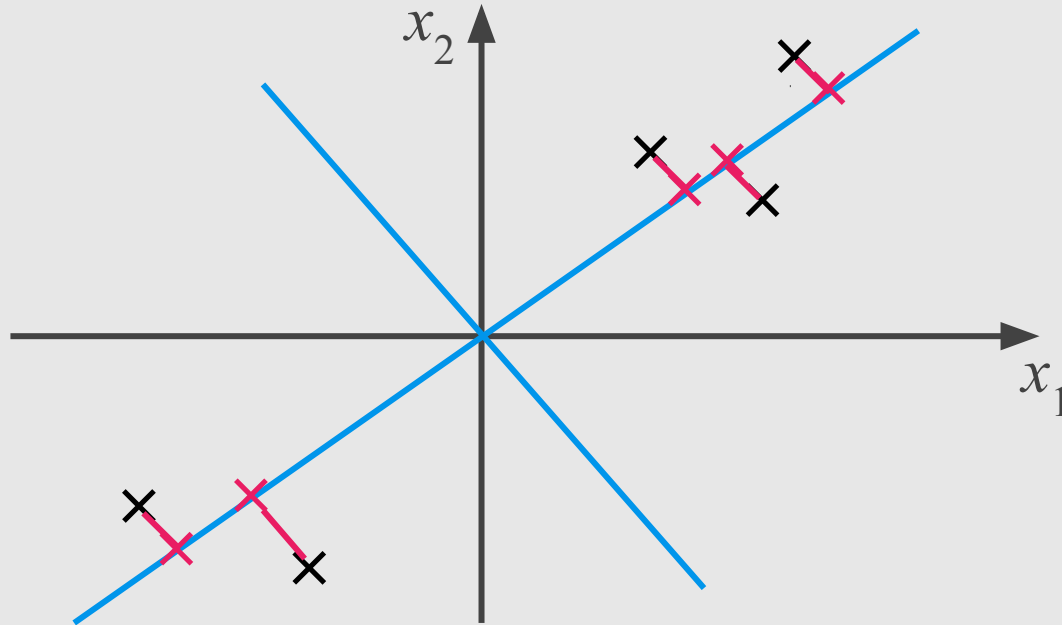
# Problem Formulation (PCA)



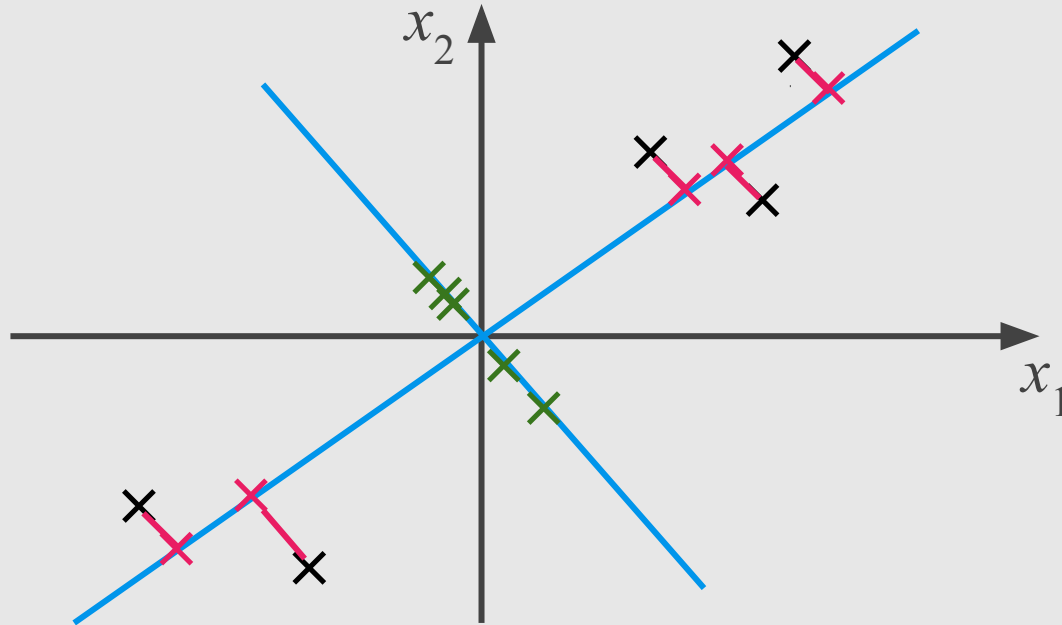
# Problem Formulation (PCA)



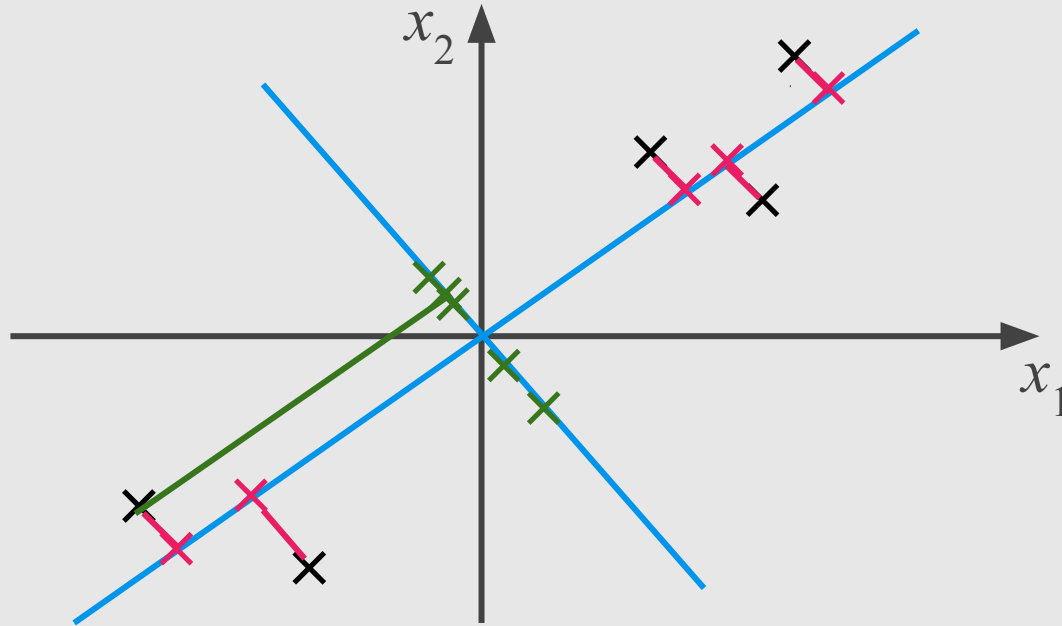
# Problem Formulation (PCA)



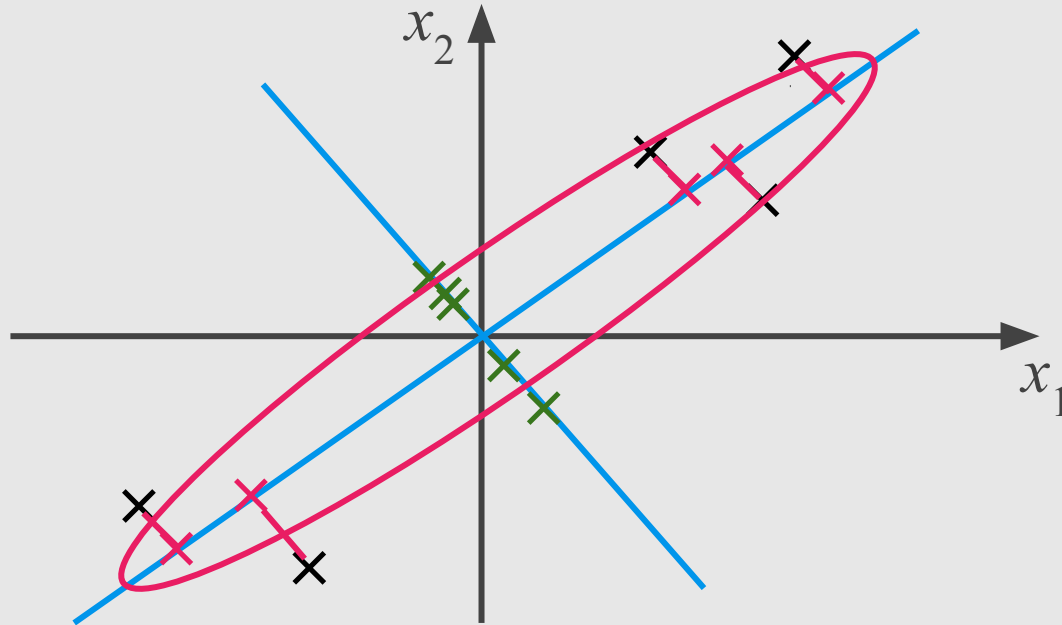
# Problem Formulation (PCA)



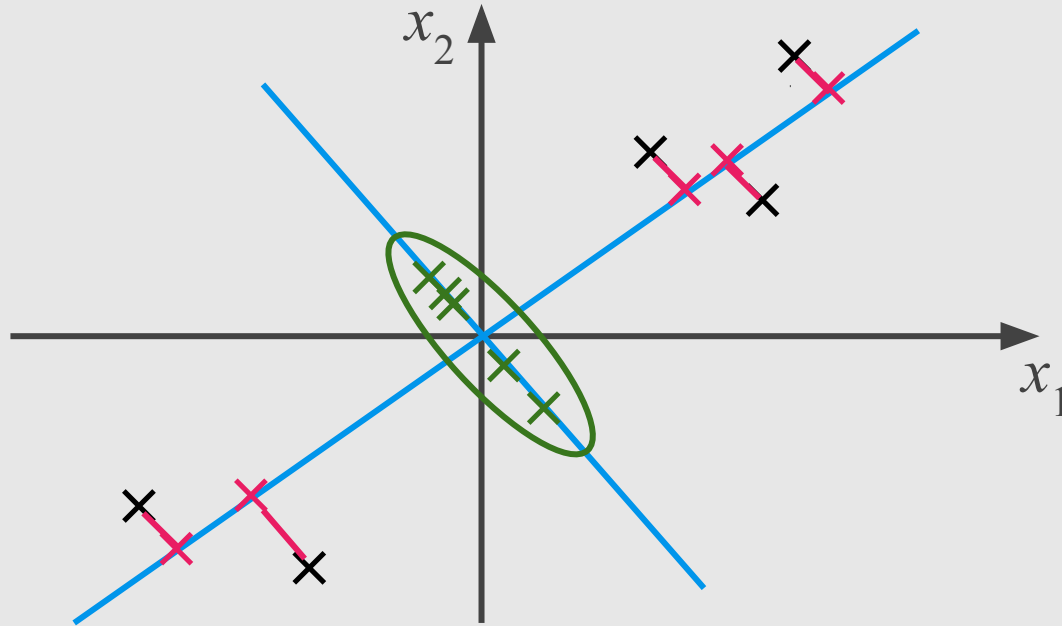
# Problem Formulation (PCA)



# Problem Formulation (PCA)



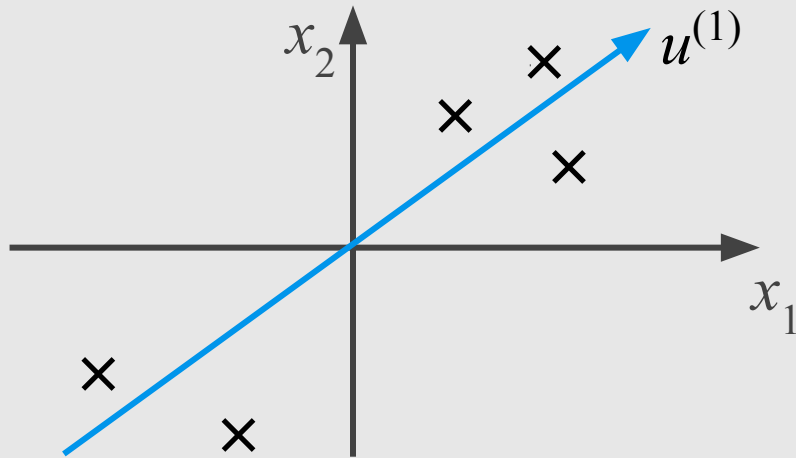
# Problem Formulation (PCA)





# 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  $u^{(1)}, u^{(2)}, \dots, u^{(k)}$  onto which to project the data, so as to minimize the projection error.

# PCA Algorithm By Eigen Decomposition

# PCA in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. Compute covariance matrix  $\Sigma$
3. Find eigenvectors  $u$  and eigenvalues  $\lambda$
4. Sort eigenvalues and pick first  $k$  eigenvectors
5. Project data to  $k$  eigenvectors

# PCA in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. Compute covariance matrix  $\Sigma$
3. Find eigenvectors  $u$  and eigenvalues  $\lambda$
4. Sort eigenvalues and pick first  $k$  eigenvectors
5. Project data to  $k$  eigenvectors

# Data Preprocessing

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

Preprocessing (feature scaling/mean normalization):

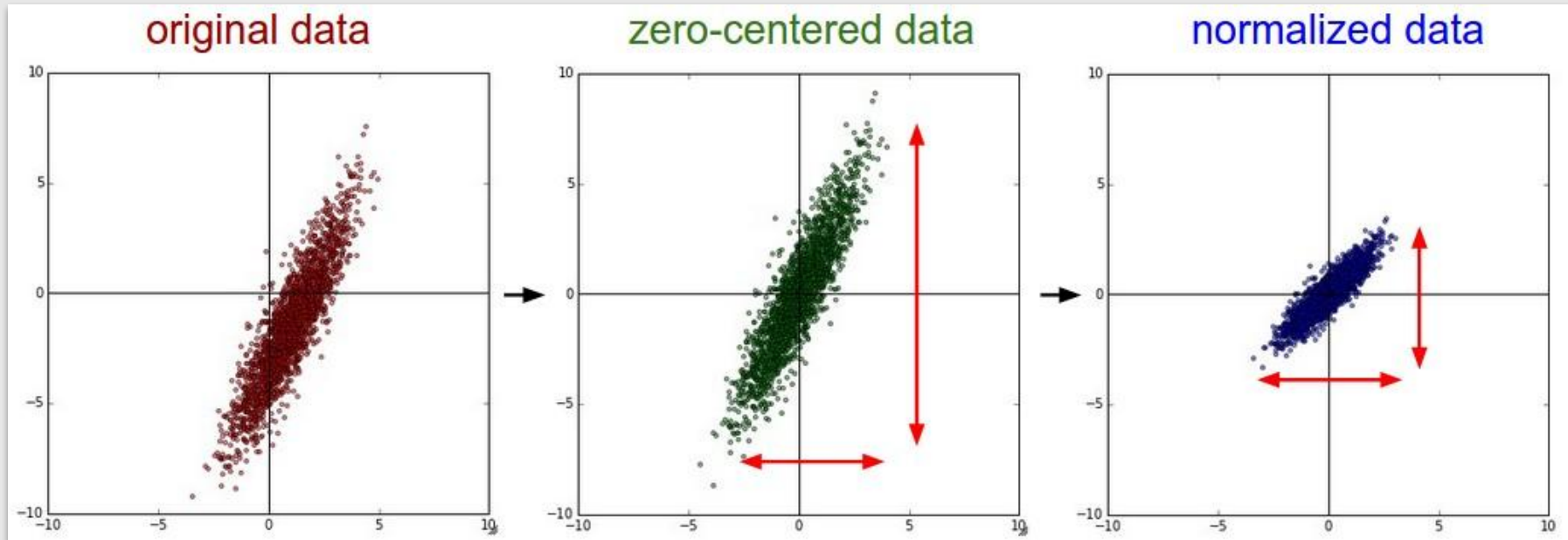
$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

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

Center the data

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

# Data Preprocessing



Credit: <http://cs231n.github.io/neural-networks-2/>

# PCA in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. **Compute covariance matrix  $\Sigma$**
3. Find eigenvectors  $u$  and eigenvalues  $\lambda$
4. Sort eigenvalues and pick first  $k$  eigenvectors
5. Project data to  $k$  eigenvectors



# PCA Algorithm

Reduce data from  $n$ -dimensions to  $k$ -dimensions

Compute “covariance matrix”:

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

# PCA Algorithm

Reduce data from  $n$ -dimensions to  $k$ -dimensions

Compute “covariance matrix”:

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

Covariance of dimensions  $x_1$  and  $x_2$ :

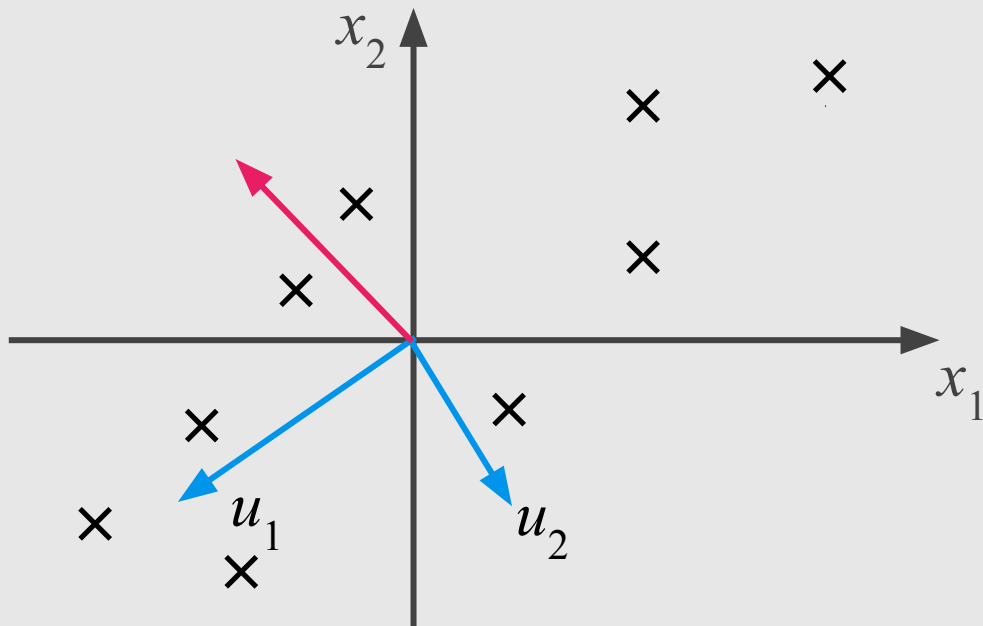
- Do  $x_1$  and  $x_2$  tend to increase together?
- or does  $x_2$  decrease as  $x_1$  increases?

$$\begin{matrix} & x_1 & x_2 \\ x_1 & \begin{bmatrix} 2.0 & 0.8 \end{bmatrix} \\ x_2 & \begin{bmatrix} 0.8 & 0.6 \end{bmatrix} \end{matrix}$$

# PCA Algorithm

Multiply a vector by  $\Sigma$  :

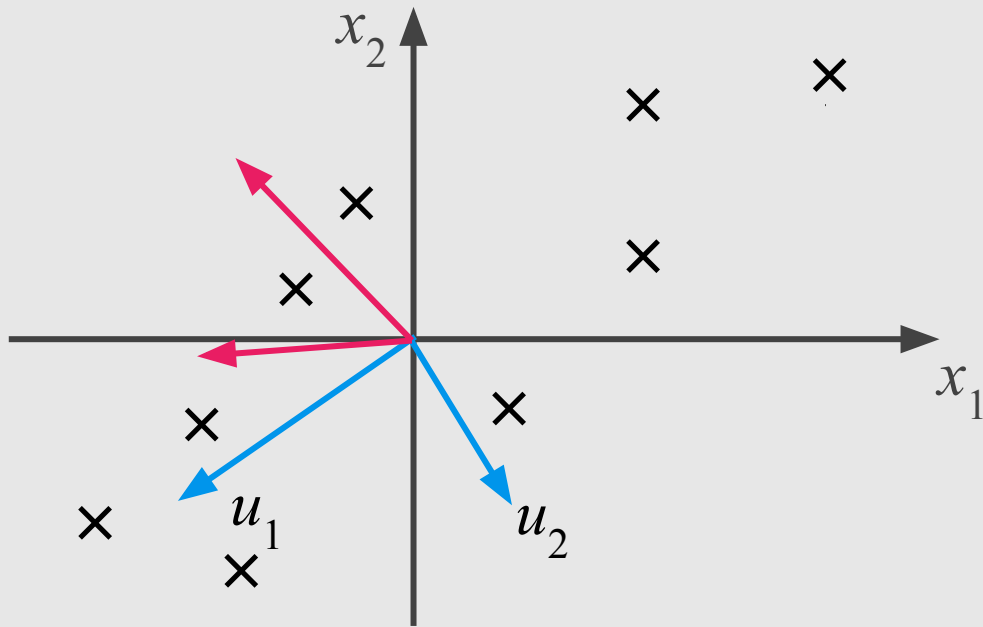
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$



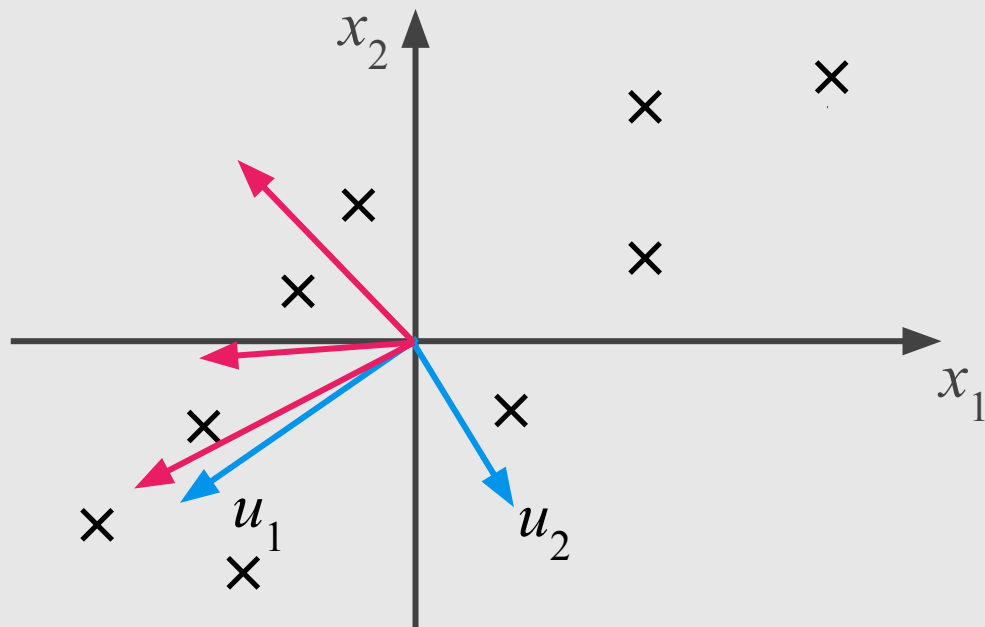
# PCA Algorithm

Multiply a vector by  $\Sigma$  :

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$



# PCA Algorithm

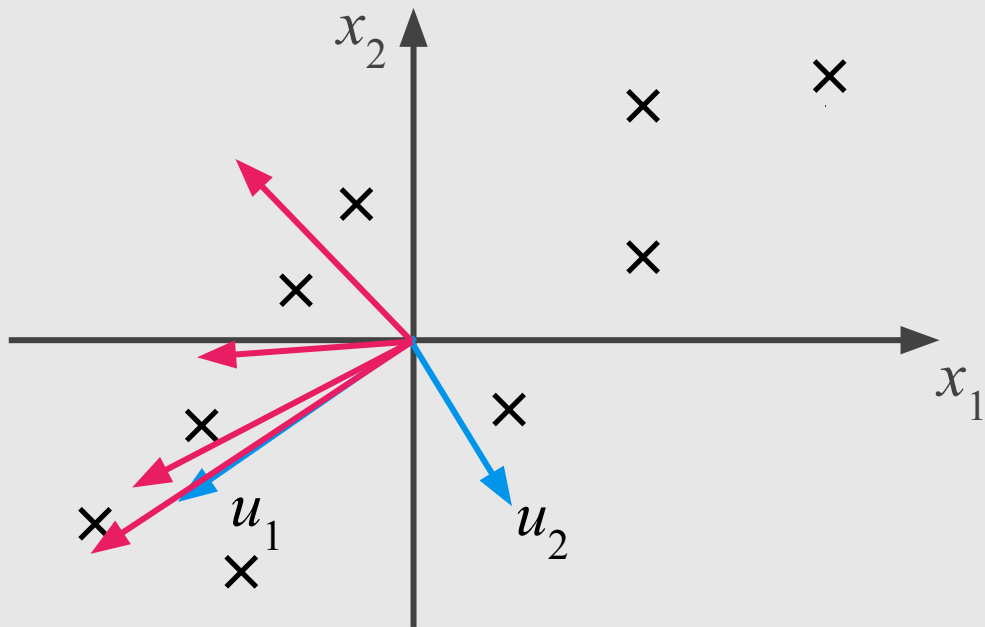


Multiply a vector by  $\Sigma$  :

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix}$$

# PCA Algorithm



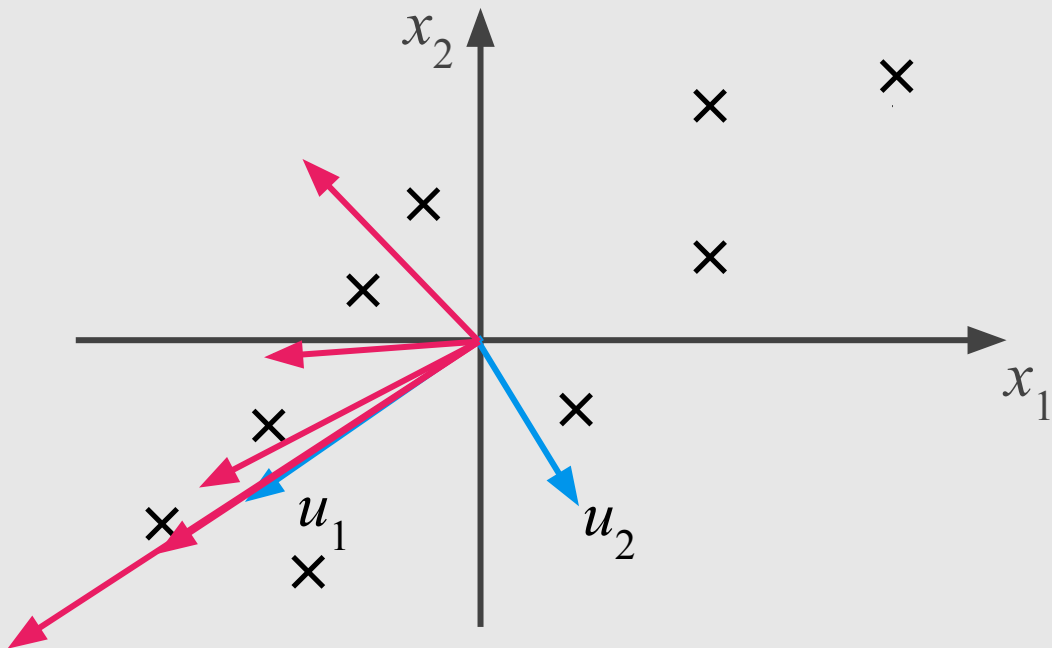
Multiple a vector by  $\Sigma$  :

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} -6.0 \\ -2.7 \end{bmatrix}$$

# PCA Algorithm



Multiple a vector by  $\Sigma$  :

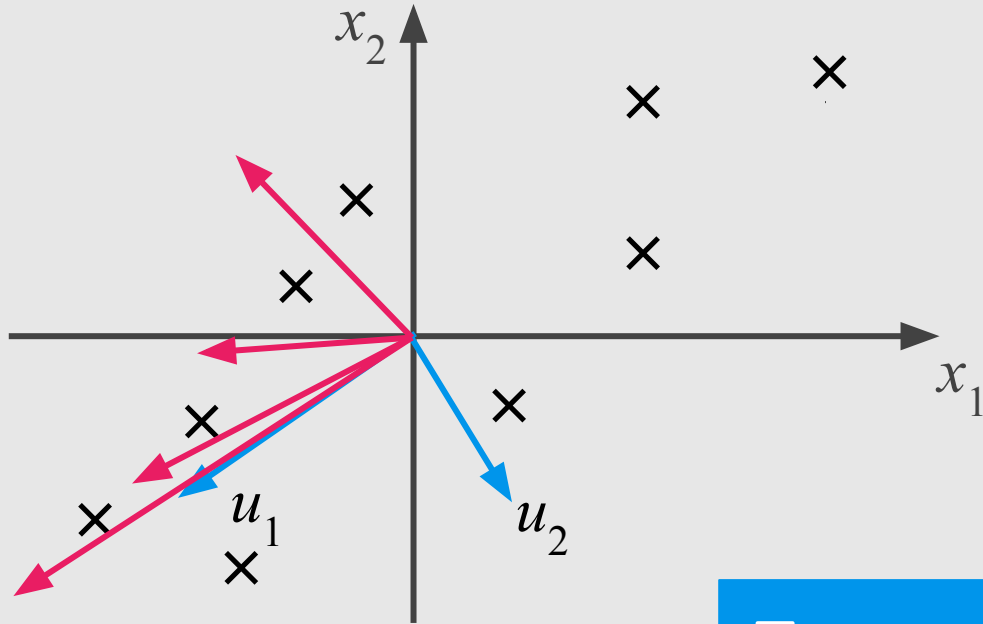
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} -6.0 \\ -2.7 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -6.0 \\ -2.7 \end{bmatrix} = \begin{bmatrix} -14.1 \\ -6.4 \end{bmatrix}$$

# PCA Algorithm



Multiple a vector by  $\Sigma$  :

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} -6.0 \\ -2.7 \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -6.0 \\ -2.7 \end{bmatrix} = \begin{bmatrix} -14.1 \\ -6.4 \end{bmatrix}$$

Turns towards direction of variation

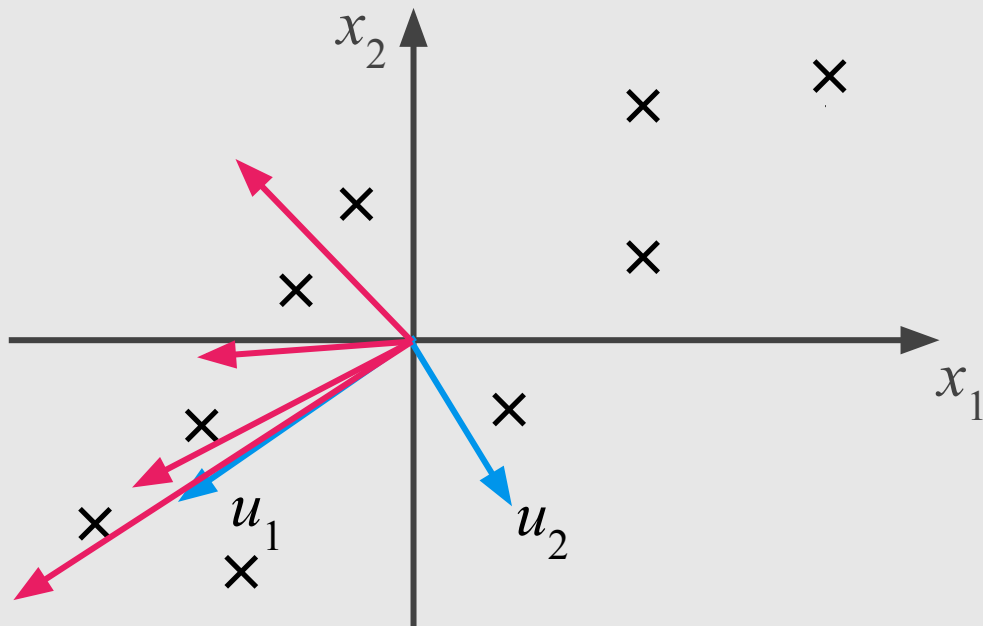


# PCA Algorithm

Want vectors  $u$  which aren't turned:  $\Sigma u = \lambda u$

$u$  = eigenvectors of  $\Sigma$

$\lambda$  = eigenvalues

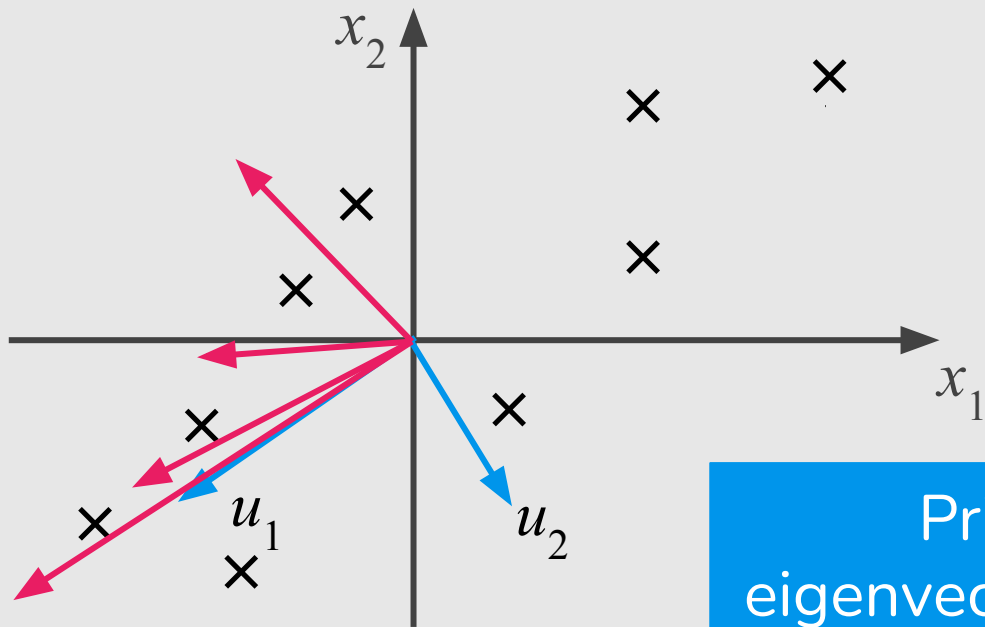


# PCA Algorithm

Want vectors  $u$  which aren't turned:  $\Sigma u = \lambda u$

$u$  = eigenvectors of  $\Sigma$

$\lambda$  = eigenvalues



Principal components =  
eigenvectors w. largest eigenvalues

# PCA in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. Compute covariance matrix  $\Sigma$
3. Find eigenvectors  $u$  and eigenvalues  $\lambda$
4. Sort eigenvalues and pick first  $k$  eigenvectors
5. Project data to  $k$  eigenvectors

# Finding Principal Components

1. Find eigenvalues by solving:  $\det(\mathbf{\Sigma} - \lambda\mathbf{I}) = 0$

$$\det \begin{bmatrix} 2.0-\lambda & 0.8 \\ 0.8 & 0.6-\lambda \end{bmatrix} = (2.0-\lambda)(0.6-\lambda) - (0.8)(0.8) = \lambda^2 - 2.6\lambda + 0.56 = 0$$

$$\{\lambda_1, \lambda_2\} = \{2.36, 0.23\}$$

# Finding Principal Components

2. Find  $i^{\text{th}}$  eigenvector by solving:  $\Sigma u_i = \lambda_i u_i$

# Finding Principal Components

2. Find  $i^{\text{th}}$  eigenvector by solving:  $\Sigma u_i = \lambda_i u_i$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} = 2.36 \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} \Rightarrow \begin{cases} 2.0u_{11} + 0.8u_{12} = 2.36u_{11} \\ 0.8u_{11} + 0.6u_{12} = 2.36u_{12} \end{cases} \Rightarrow u_{11} = 2.2u_{12}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} u_{21} \\ u_{22} \end{bmatrix} = 0.23 \begin{bmatrix} u_{21} \\ u_{22} \end{bmatrix} \Rightarrow u_2 = \begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix}$$

$$u_1 \sim \begin{bmatrix} 2.2 \\ 1 \end{bmatrix}$$

Want  $\|u_1\|=1$

3. 1<sup>st</sup> PC:  $\begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}$  and 2<sup>nd</sup> PC:  $\begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix}$

$$\begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}$$

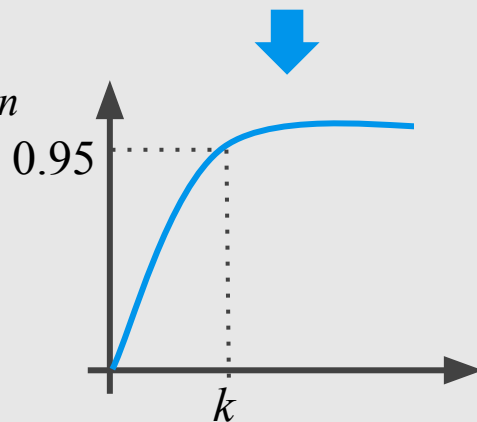
# PCA in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. Compute covariance matrix  $\Sigma$
3. Find eigenvectors  $u$  and eigenvalues  $\lambda$
4. **Sort eigenvalues and pick first  $k$  eigenvectors**
5. Project data to  $k$  eigenvectors

# How many PCs?

- Have eigenvectors  $u_1, u_2, \dots, u_n$ , want  $k < n$
- eigenvalue  $\lambda_i$  = variance along  $u_i$
- Pick  $u_i$  that explain the most variance:
  - Sort eigenvectors s.t.  $\lambda_1 > \lambda_2 > \lambda_3 > \dots > \lambda_n$
  - Pick first  $k$  eigenvectors which explain 95% of total variance
    - Typical threshold: 90%, 95%, 99%

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^n \lambda_i} \leq 1$$





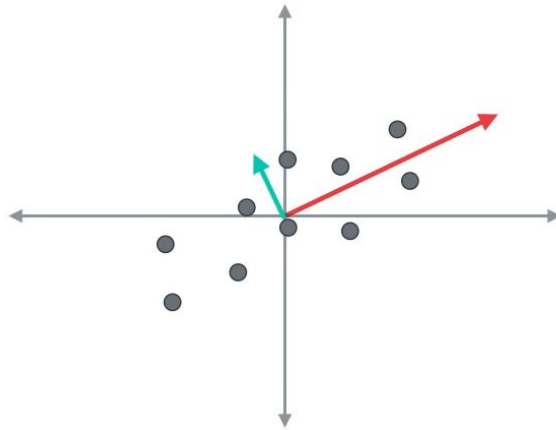
# PCA in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. Compute covariance matrix  $\Sigma$
3. Find eigenvectors  $u$  and eigenvalues  $\lambda$
4. Sort eigenvalues and pick first  $k$  eigenvectors
5. **Project data to  $k$  eigenvectors**

# Principal Component Analysis (1 video, 26 min), Luis Serrano

<https://youtu.be/g-Hb26agBFg>

## Principal Component Analysis (PCA)



$$\Sigma = \begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

11

$$\begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

1

Eigenvectors  
(direction)

Eigenvalues  
(magnitude)

# Principal Component Analysis (12 videos, 3-15 min)

[https://www.youtube.com/playlist?list=PLBu09BD7ez\\_5\\_yapAg86Od6JeeypkS4YM](https://www.youtube.com/playlist?list=PLBu09BD7ez_5_yapAg86Od6JeeypkS4YM)

Search

**Curse of dimensionality**

- Datasets typically high dimensional
  - vision:  $10^4$  pixels, text:  $10^6$  words
    - the way we observe / record them
  - true dimensionality often much lower
    - a manifold (sheet) in a high-d space
- Example: handwritten digits
  - 28 x 28 bitmap:  $\{0,1\}^{400}$  possible events
    - will never see most of these events
  - actual digits: tiny fraction of events
  - true dimensional

PLAY ALL

## Principal Component Analysis

12 videos • 119,895 views • Last updated on May 21, 2014



Victor Lavrenko

SUBSCRIBE 19K

Lectures 18 and 19 in the Introductory Applied Machine Learning (IAML) course by Victor Lavrenko at the

1

**Curse of dimensionality**

- Datasets typically high dimensional
  - vision:  $10^4$  pixels, text:  $10^6$  words
  - the way we observe / record them
- Example: handwritten digits
  - 28 x 28 bitmap:  $\{0,1\}^{400}$  possible events
  - will never see most of these events
  - actual digits: tiny fraction of events
  - true dimensionality

10:00

### PCA 1: curse of dimensionality

Victor Lavrenko

2

**Learning with high dimensionality**

- Use domain knowledge
  - feature engineering, DTF, MFC
- Make assumption about dimensions
  - independent, correlated
  - dimensionality reduction
  - sparse or regularized regression
  - sparsity: e.g. maximum number of dimensions  $k \ll D$

6:06

### PCA 2: dimensionality reduction

Victor Lavrenko

3

**Very greatest variance?**

- Example: reduce 2-dimensional data to 1-d
  - $(x_1, x_2) \rightarrow x$  (along new axis)
- Pick  $x$  to maximize variability
- Reduce cases when two points are close in  $x$ -space but very far in  $(x_1, x_2)$ -space
- Minimize distances between original points

5:32

### PCA 3: direction of greatest variance

Victor Lavrenko

4

**Principal components**

- "Center" the data at origin:  $x_i = x_i - \bar{x}$ 
  - subtract mean from each attribute
- Compute covariance matrix  $S$ 
  - covariance of dimensions  $x_i$  and  $x_j$
  - $S_{ij} = \frac{1}{n} \sum_{k=1}^n (x_i - \bar{x}_i)(x_j - \bar{x}_j)$
  - $S$  is symmetric,  $S = S^T$
  - $S$  is dense, symmetric,  $n \times n$
- Multiply a vector  $\mathbf{v}$  by  $S$ :  $\mathbf{w} = S\mathbf{v}$ 
  - turns towards direction of variance

6:58

### PCA 4: principal components = eigenvectors

Victor Lavrenko

5

**Finding principal components**

1. Find eigenvalues by solving:  $\det(S - \lambda I) = 0$ 
  - $S = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix}$ ,  $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ ,  $\lambda = \text{eigenvalue}$
  - $\det(S - \lambda I) = (S_{11} - \lambda)(S_{22} - \lambda) - S_{12}S_{21} = 0$
  - $\lambda^2 - (S_{11} + S_{22})\lambda + (S_{11}S_{22} - S_{12}S_{21}) = 0$
  - $\lambda = \frac{(S_{11} + S_{22}) \pm \sqrt{(S_{11} - S_{22})^2 + 4S_{12}S_{21}}}{2}$
2. Find  $\mathbf{P}$ -eigenvector by solving:  $S\mathbf{p} = \lambda\mathbf{p}$ 
  - $\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$ ,  $\mathbf{p}^T \mathbf{p} = 1$ ,  $\mathbf{p} \perp \mathbf{q}$ ,  $\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$
  - $\mathbf{p} = \frac{1}{\sqrt{p_1^2 + p_2^2}} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$
  - $\mathbf{q} = \frac{1}{\sqrt{q_1^2 + q_2^2}} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$

5:03

### PCA 5: finding eigenvalues and eigenvectors

Victor Lavrenko



Search



- Home
- Trending
- Subscriptions

LIBRARY

- History
- Watch later
- Liked videos
- Neural Networks ...



## Essence of linear algebra

14 videos • 3,671,987 views • Last updated on Aug 1, 2018



3Blue1Brown

A geometric understanding of matrices, determinants, eigen-stuffs and more.

10



3BLUE1BROWN SERIES S1 • E10

Cross products | Essence of linear algebra, Chapter 10

3Blue1Brown

11

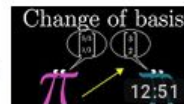


3BLUE1BROWN SERIES S1 • E11

Cross products in the light of linear transformations | Essence of linear algebra

3Blue1Brown

12

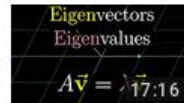


3BLUE1BROWN SERIES S1 • E12

Change of basis | Essence of linear algebra, chapter 12

3Blue1Brown

13

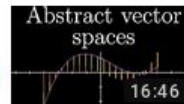


3BLUE1BROWN SERIES S1 • E13

Eigenvectors and eigenvalues | Essence of linear algebra, chapter 13

3Blue1Brown

14



3BLUE1BROWN SERIES S1 • E14

Abstract vector spaces | Essence of linear algebra, chapter 14

# 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