Machine Learning and Pattern Recognition
A High Level Overview

Prof. Anderson Rocha
(Main bulk of slides kindly provided by **Prof. Sandra Avila**)
Institute of Computing (IC/Unicamp)

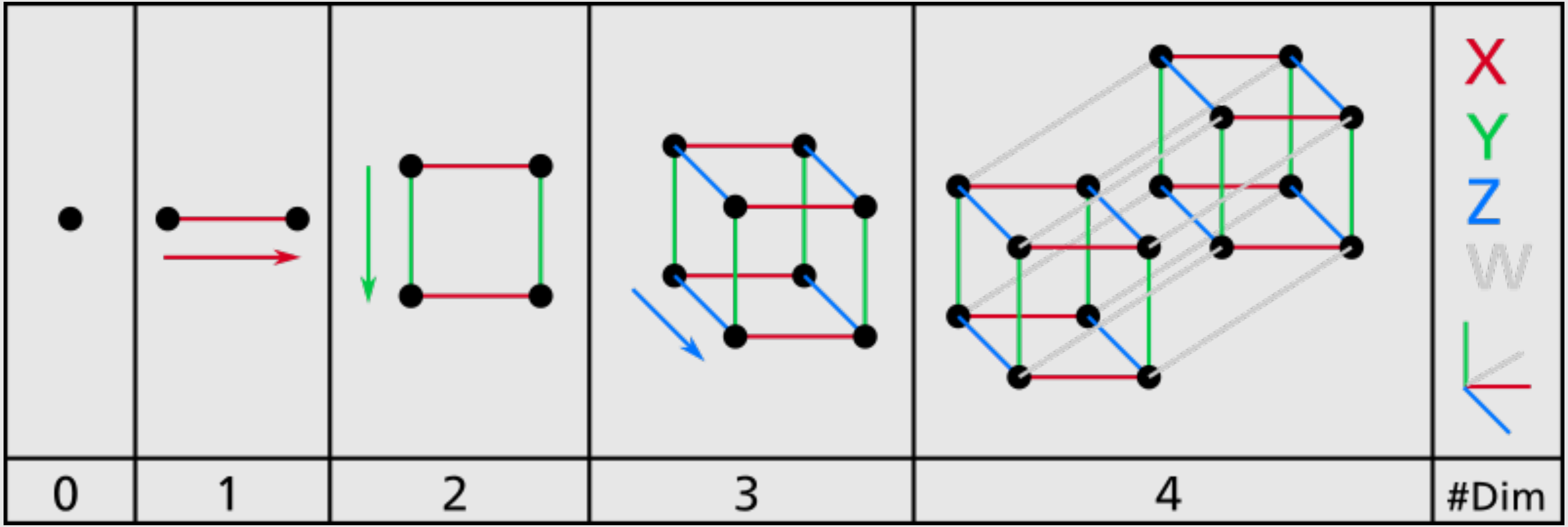
MC886/MO444

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”**

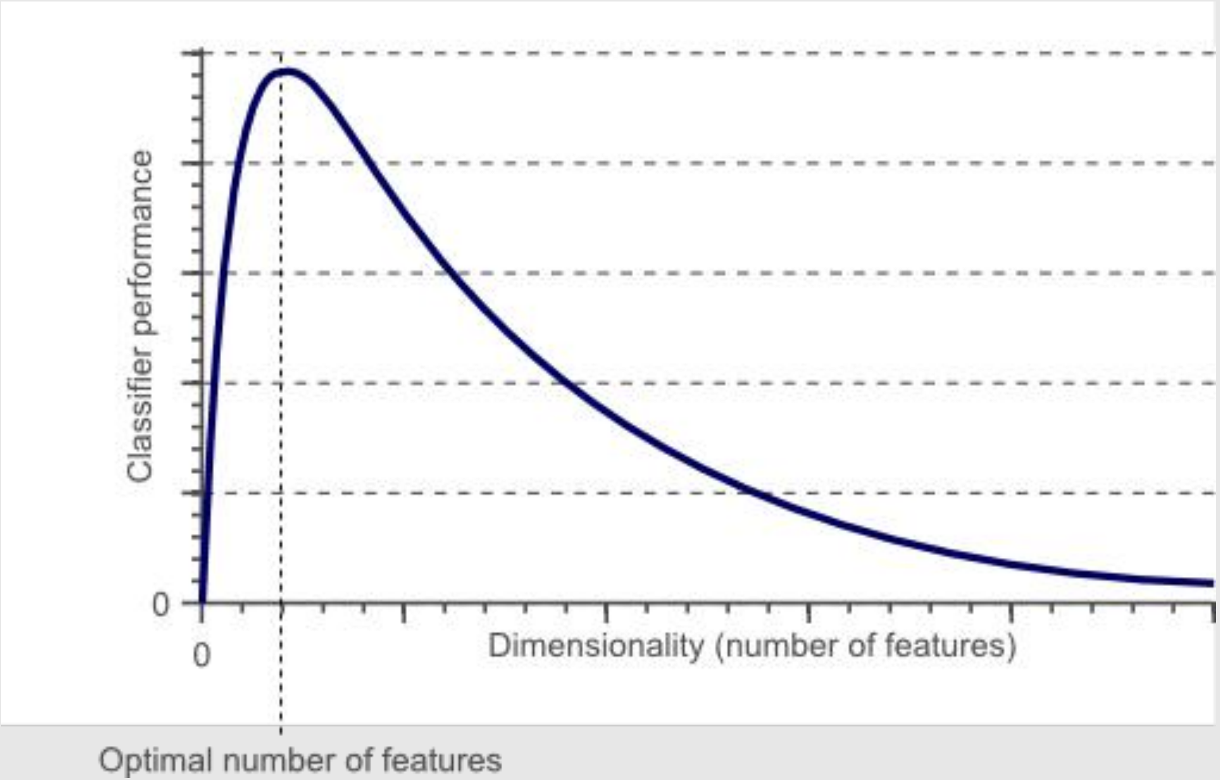
The Curse of Dimensionality

The Curse of Dimensionality



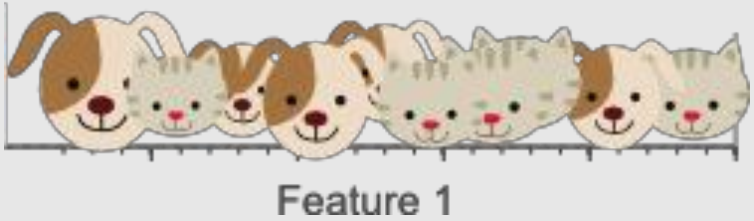
Even a basic 4D hypercube is incredibly hard to picture in our mind.

The Curse of Dimensionality



The Curse of Dimensionality

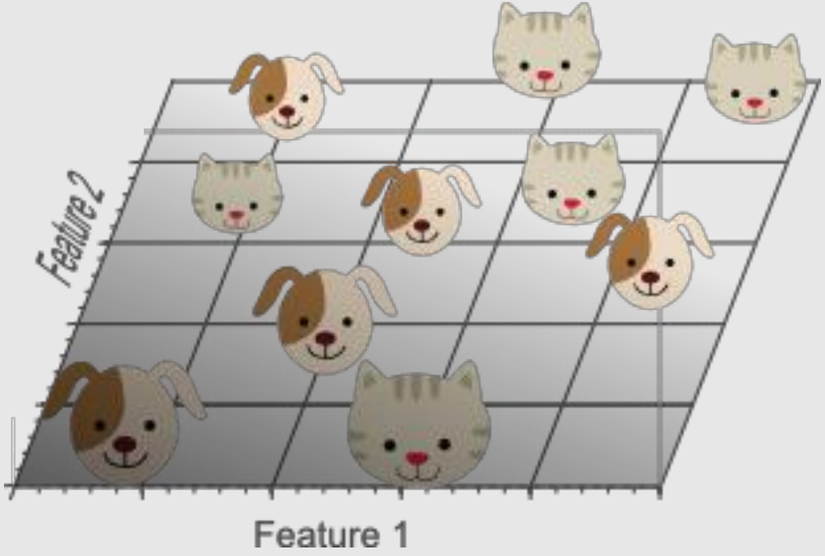
As the dimensionality of data grows, the density of observations becomes lower and lower and lower.



10 images
1 dimension: 5 regions

The Curse of Dimensionality

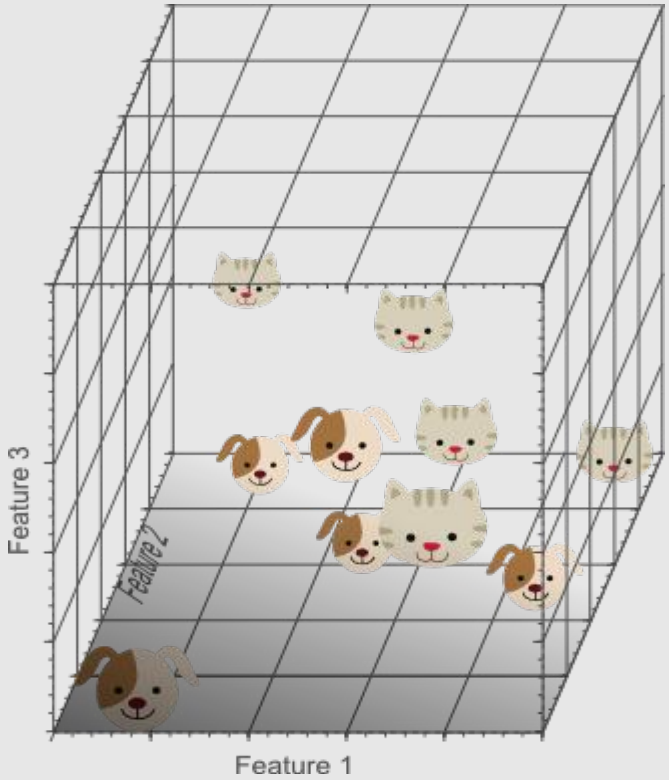
As the dimensionality of data grows, the density of observations becomes lower and lower and lower.



10 images

2 dimensions: 25 regions

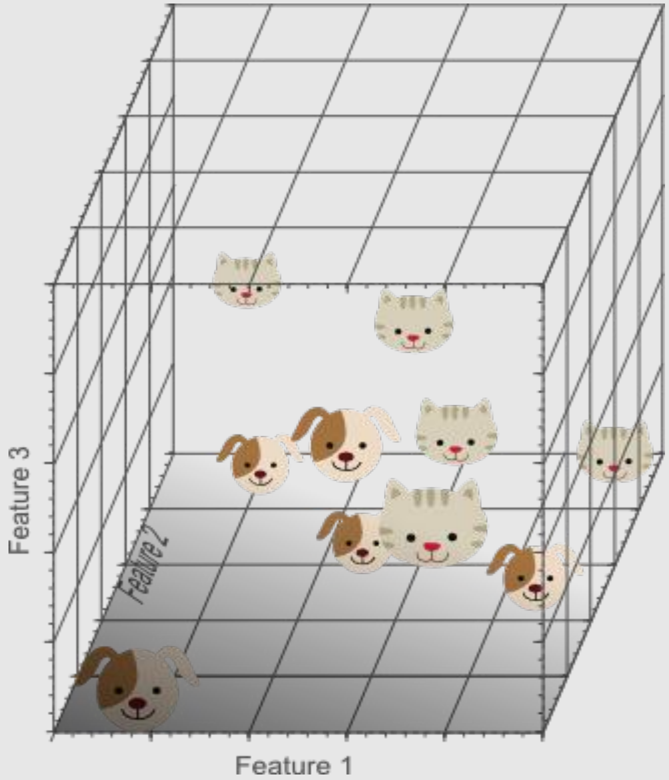
The Curse of Dimensionality



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?

- 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?

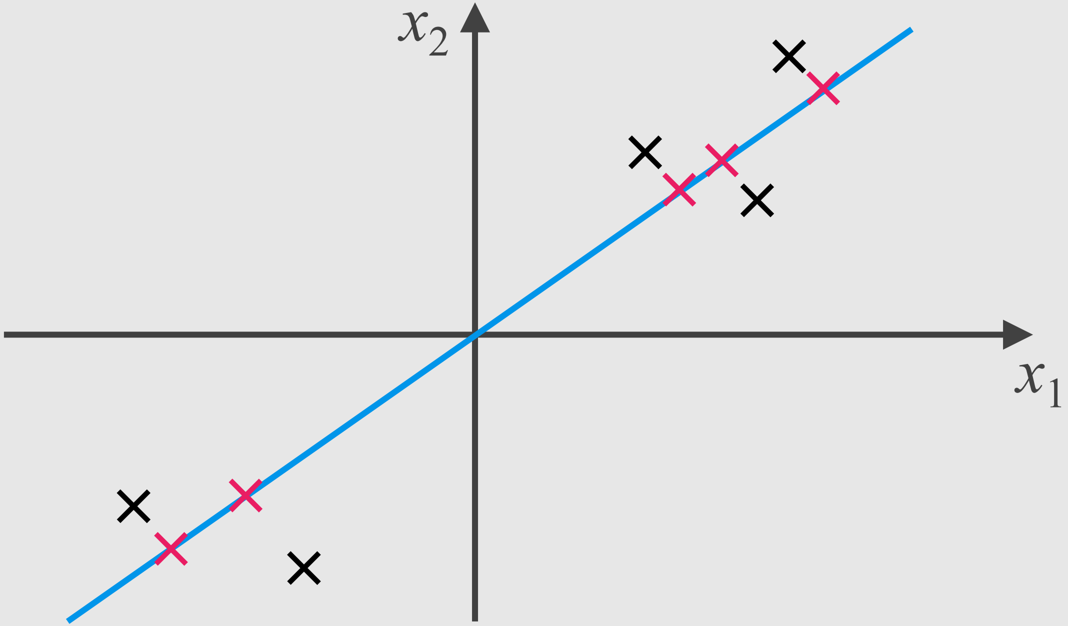
- **Feature Extraction:** create a subset of new features by combining the existing ones.
  - $z = f(x_1, x_2, x_3, x_4, x_5)$
- **Feature Selection:** choosing a subset of all the features (the ones more informative).
  - $\mathbf{x_1}, x_2, \mathbf{x_3}, x_4, \mathbf{x_5}$

PCA: Principal Component Analysis

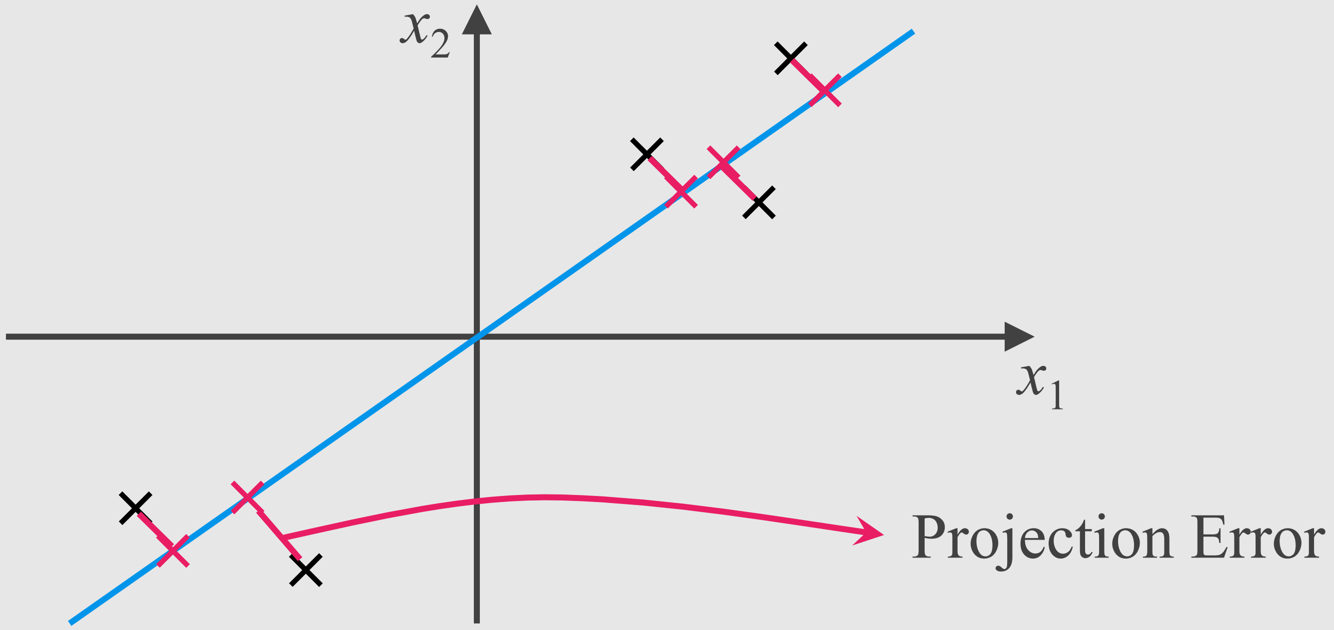
Principal Component Analysis (PCA)

- 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.

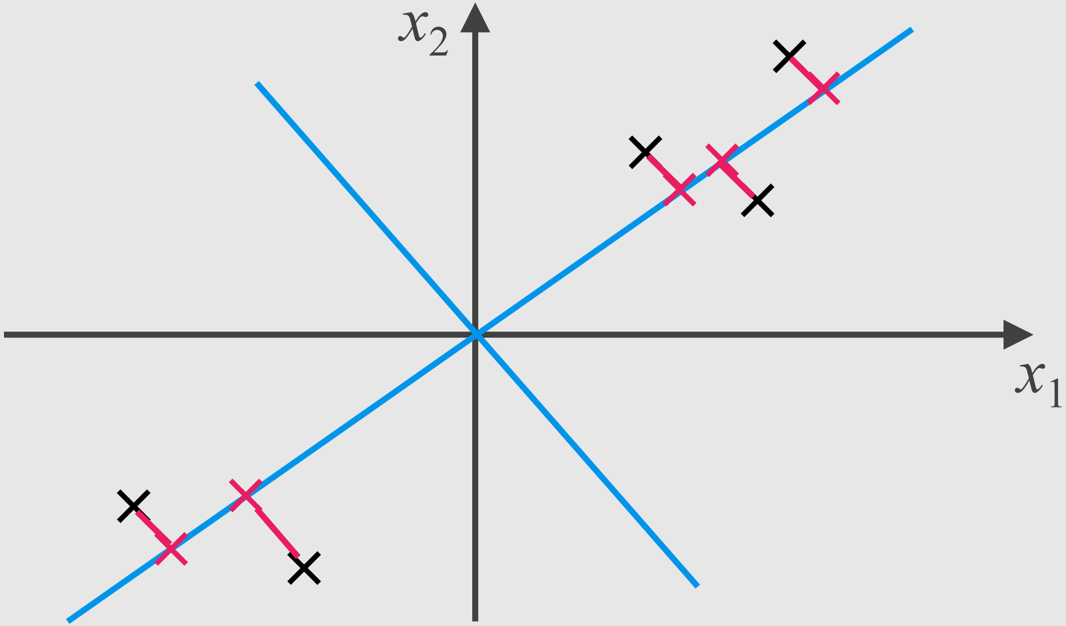
Problem Formulation (PCA)



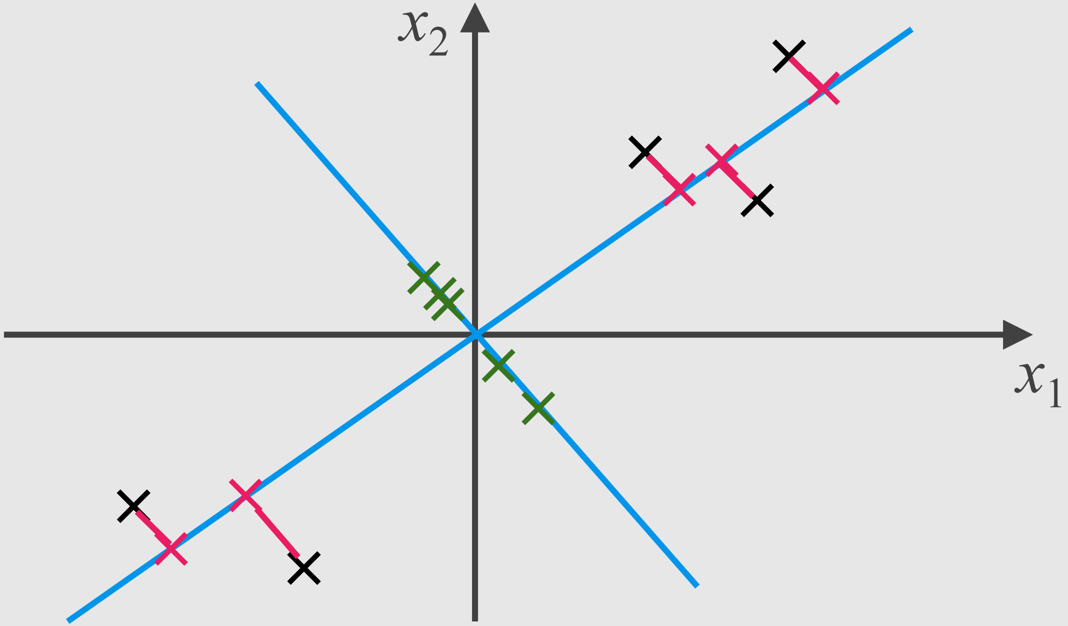
Problem Formulation (PCA)



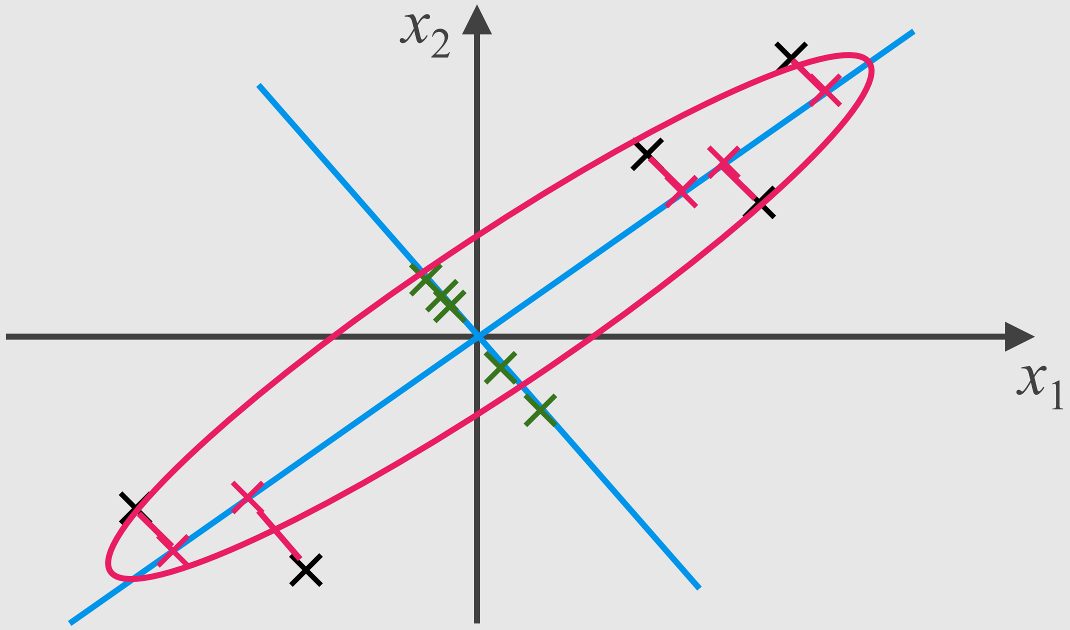
Problem Formulation (PCA)



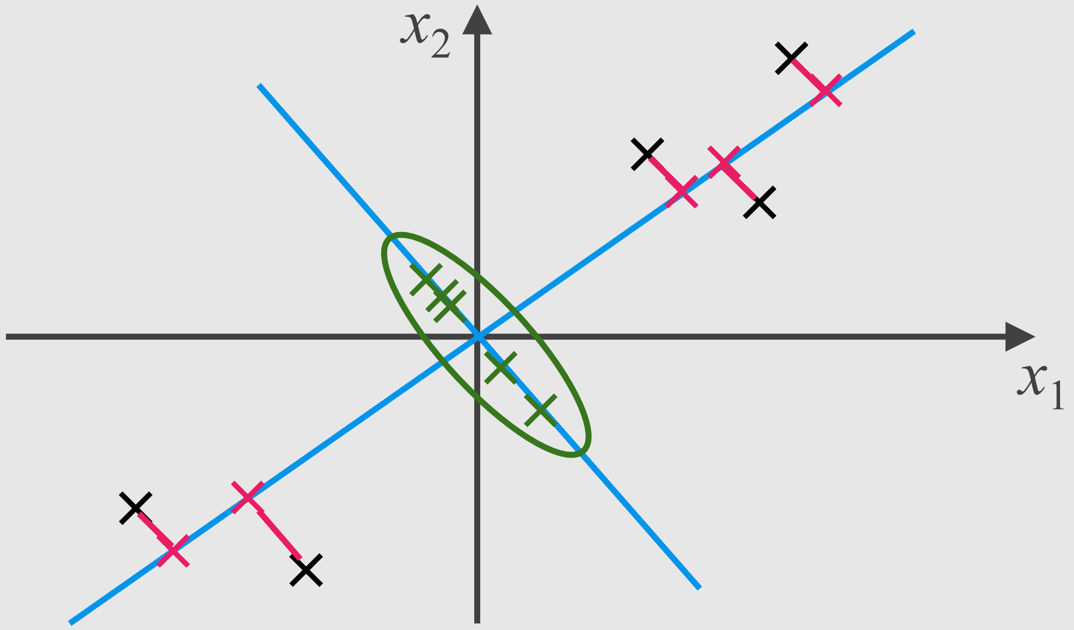
Problem Formulation (PCA)



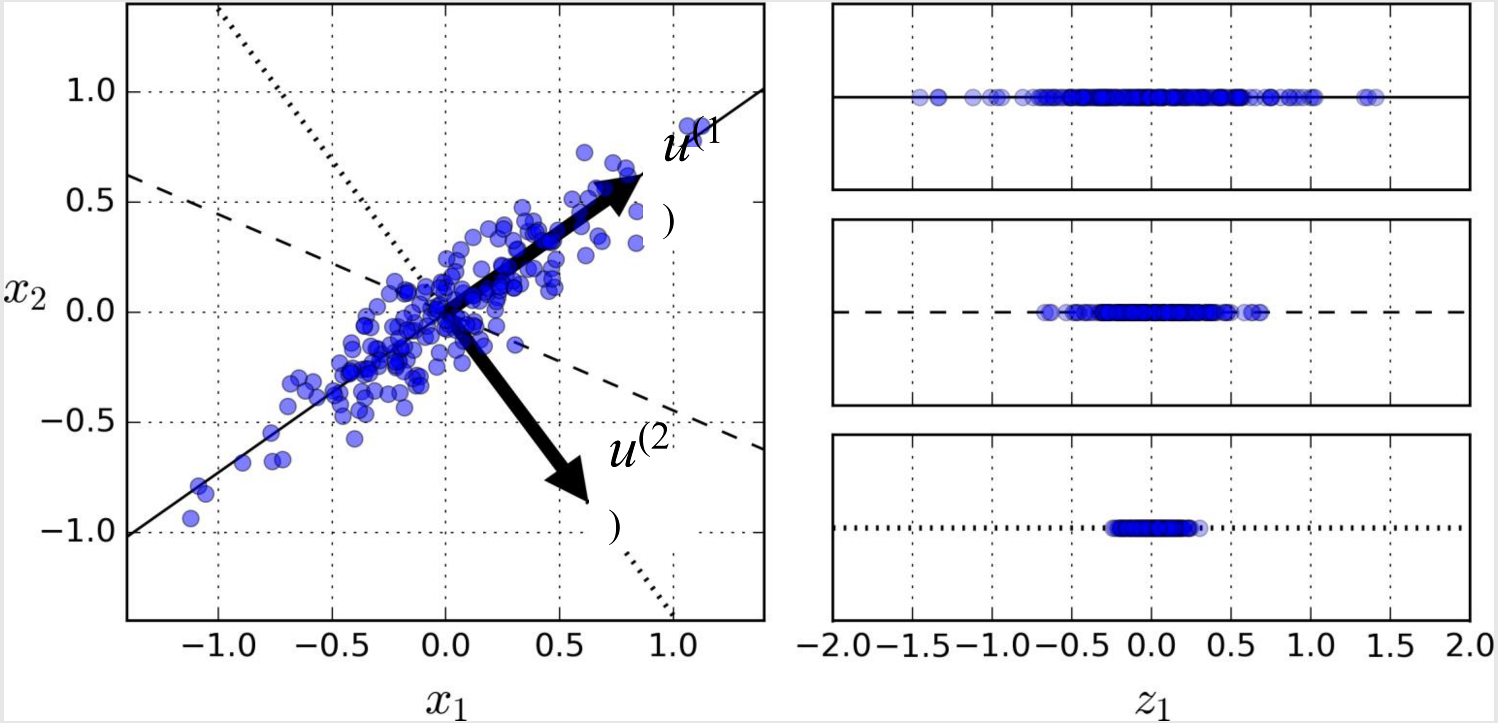
Problem Formulation (PCA)



Problem Formulation (PCA)



Preserving the Variance



PCA Algorithm
By Eigen Decomposition

Data Preprocessing

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

Preprocessing (feature scaling/mean normalization):

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

Center the data

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):

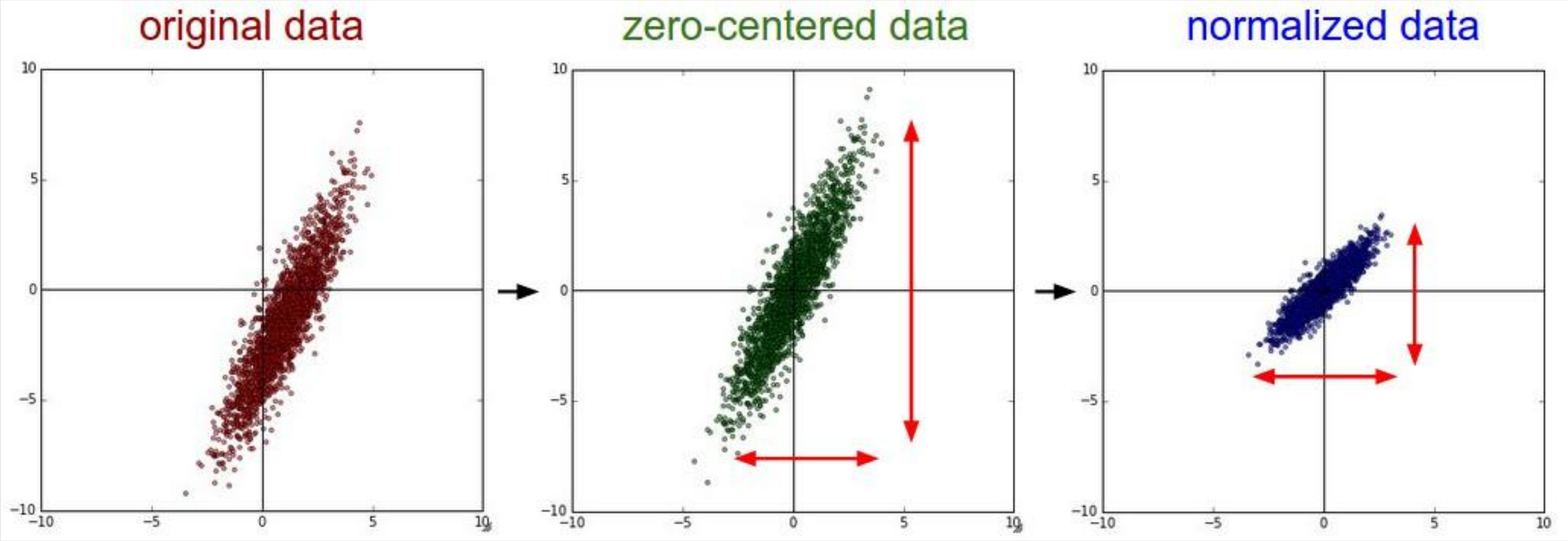
$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

Center the data

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

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 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":

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

Covariance of dimensions x_1 and x_2:

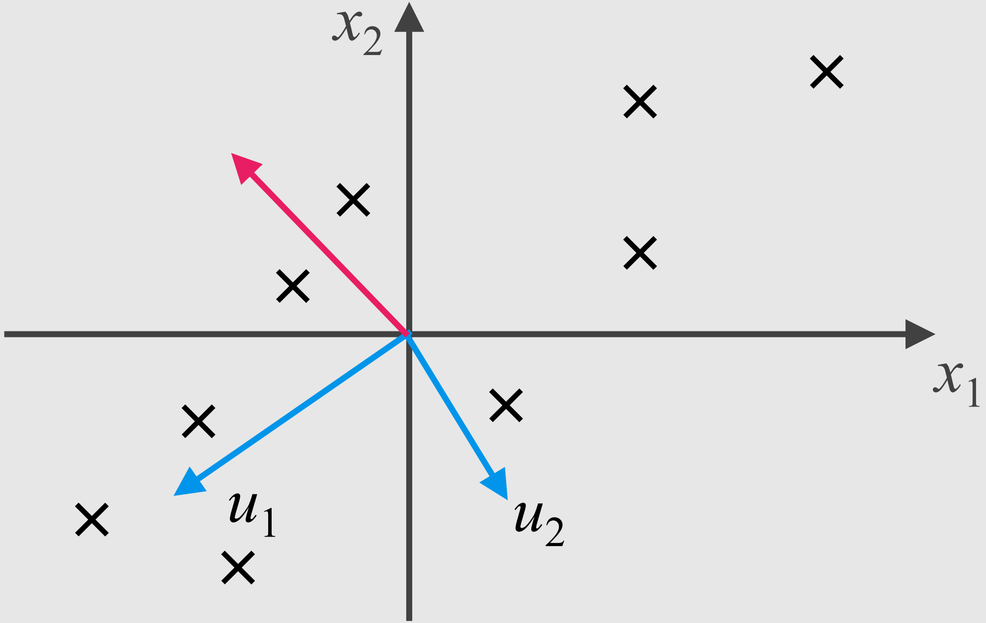
- Do x_1 and x_2 tend to increase together?
- or does x_2 decrease as x_1 increases?

$$\begin{array}{cc} & \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{array}{c} x_1 \\ x_2 \end{array} & \begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \end{array}$$

PCA Algorithm

Multiple a vector by $\boldsymbol{\Sigma}$:

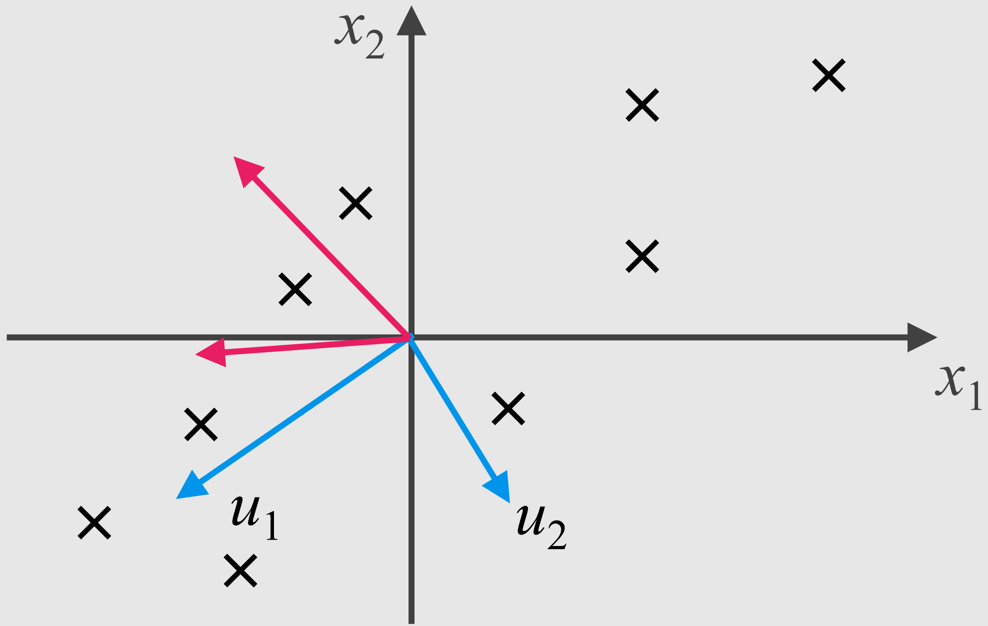
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$



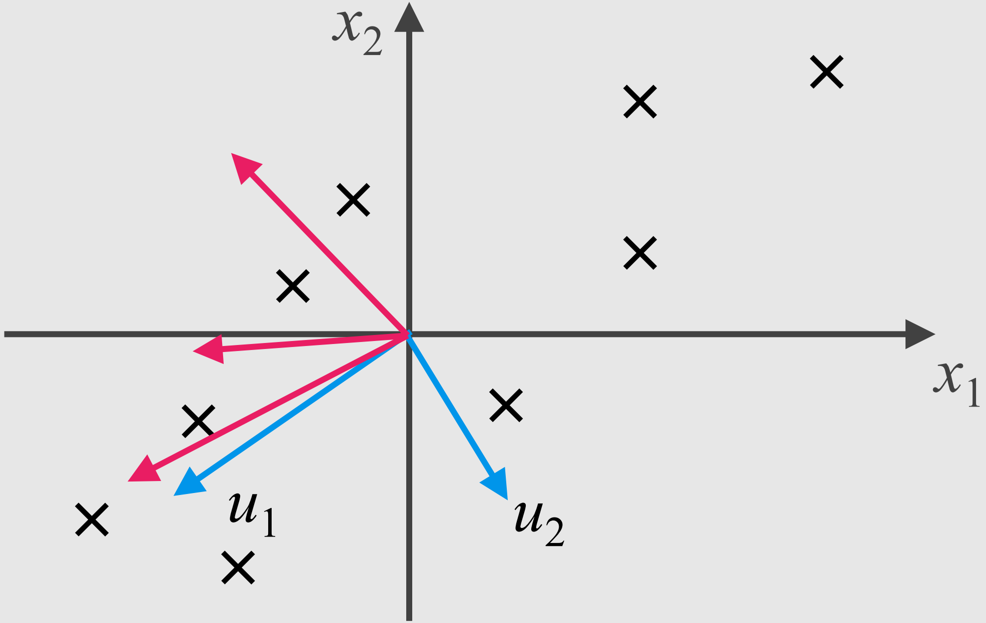
PCA Algorithm

Multiple a vector by $\boldsymbol{\Sigma}$:

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



PCA Algorithm

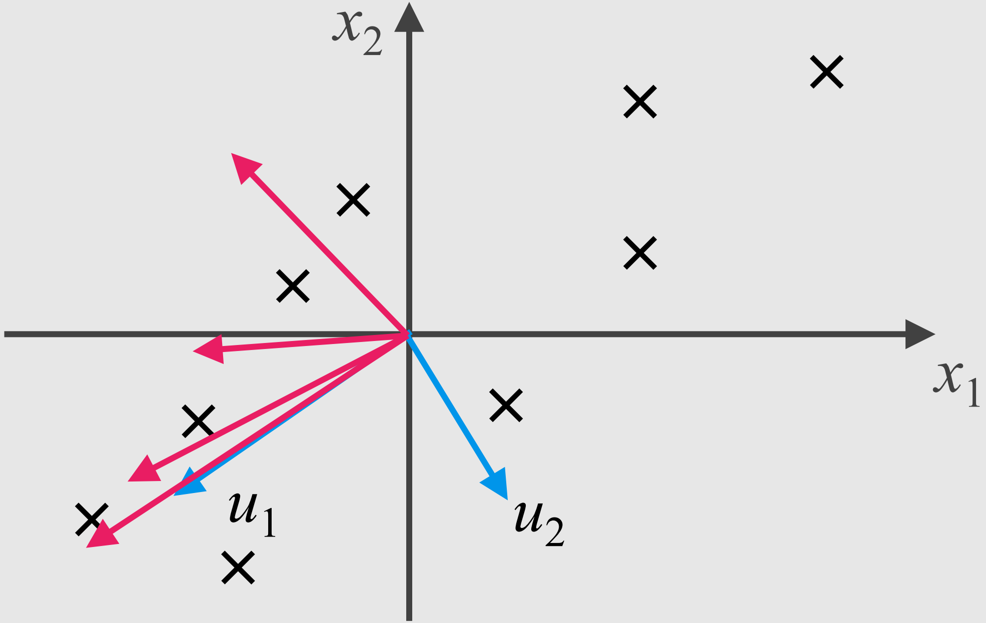


Multiple a vector by $\mathbf{\Sigma}$:

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

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

PCA Algorithm



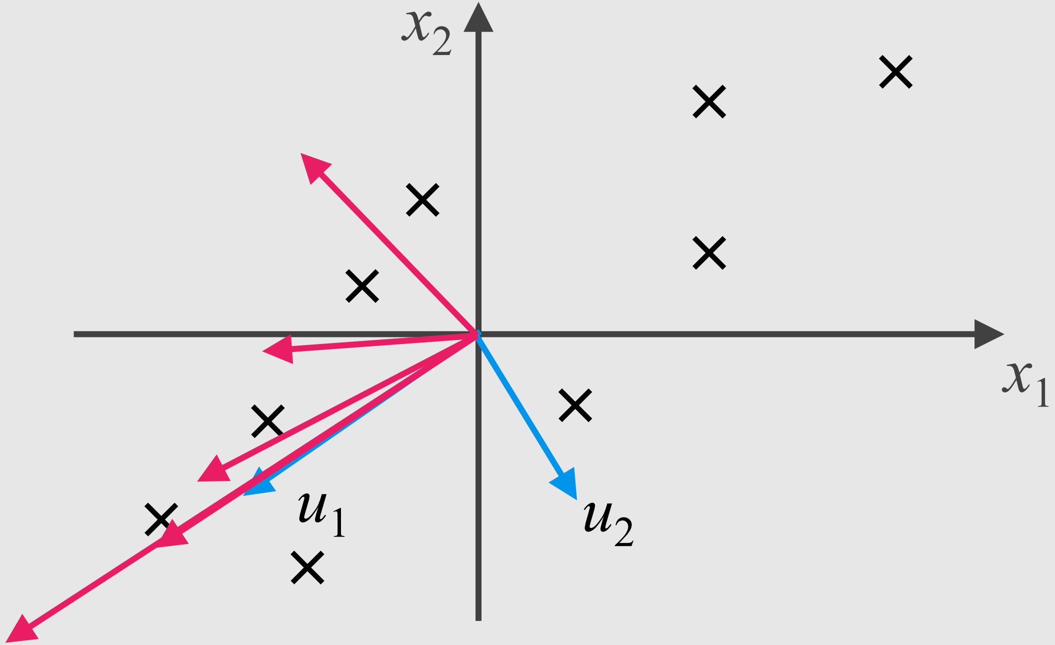
Multiple a vector by $\mathbf{\Sigma}$:

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

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

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

PCA Algorithm



Multiple a vector by $\boldsymbol{\Sigma}$:

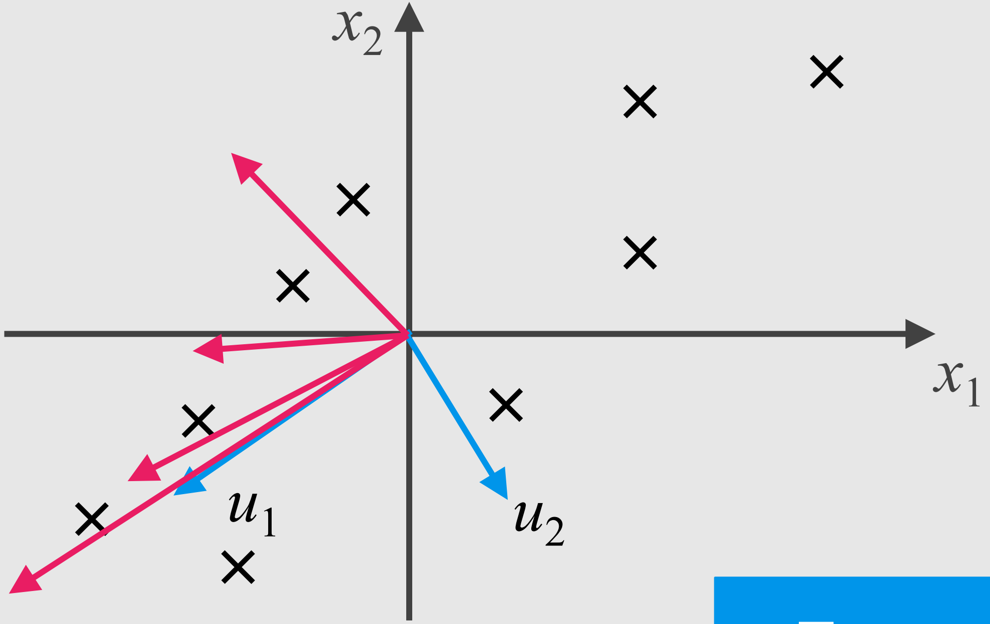
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$

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

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

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

PCA Algorithm



Multiple a vector by $\boldsymbol{\Sigma}$:

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

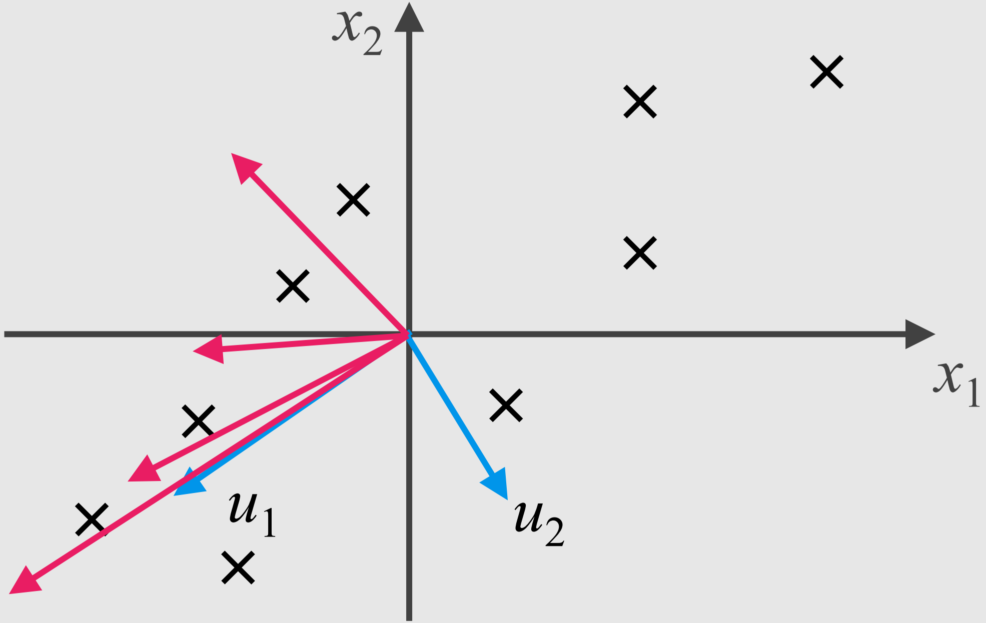
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix}$$

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

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

Turns towards direction of variation

PCA Algorithm

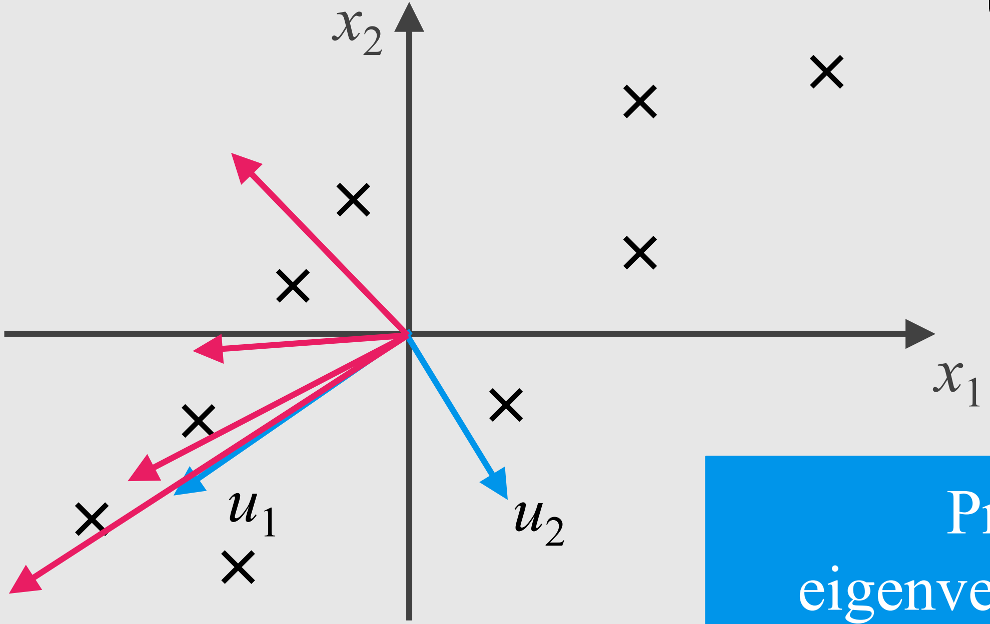


Want vectors u which aren't turned: $\boldsymbol{\Sigma} u = \lambda u$

$u =$ eigenvectors of $\boldsymbol{\Sigma}$

$\lambda =$ eigenvalues

PCA Algorithm



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

u = eigenvectors of $\mathbf{\Sigma}$

λ = eigenvalues

Principal components = eigenvectors w. largest eigenvalues

Finding Principal Components

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

$$\det\begin{bmatrix} 2.0-\lambda & 0.8 \\ 0.8 & 0.6-\lambda \end{bmatrix} =$$

Finding Principal Components

1. Find eigenvalues by solving: det(**Σ** - λI) = 0

$$\det\begin{bmatrix} 2.0-\lambda & 0.8 \\ 0.8 & 0.6-\lambda \end{bmatrix} = (2.0-\lambda)(0.6-\lambda)-(0.8)(0.8)$$

Finding Principal Components

1. Find eigenvalues by solving: det(**Σ** - λI) = 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^{th} eigenvector by solving: $\mathbf{\Sigma} u_i = \lambda_i u_i$

Finding Principal Components

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

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

Finding Principal Components

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

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

Finding Principal Components

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

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

Finding Principal Components

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

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

$$\Downarrow$$

$$\mathrm{u}_1 \sim \begin{bmatrix} 2.2 \\ 1 \end{bmatrix}$$

Finding Principal Components

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

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

$$\Downarrow$$

$$\mathrm{u}_1 \sim \begin{bmatrix} 2.2 \\ 1 \end{bmatrix}$$

$\Downarrow$ Want $||u_1||$=1

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

Finding Principal Components

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

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

$$\Downarrow$$

$$\mathrm{u}_1 \sim \begin{bmatrix} 2.2 \\ 1 \end{bmatrix}$$

$\Downarrow$ Want $||u_1||=1$

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

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

Finding Principal Components

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

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

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

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

Want $||u_1||=1$

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

2. 1^{st} PC: $\begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}$ and 2^{nd} PC: $\begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix}$

How many PCs?

- Have eigenvectors $u_1, u_2, \ldots, u_n$, want $k < n$
- eigenvalue λ_i = variance along u_i

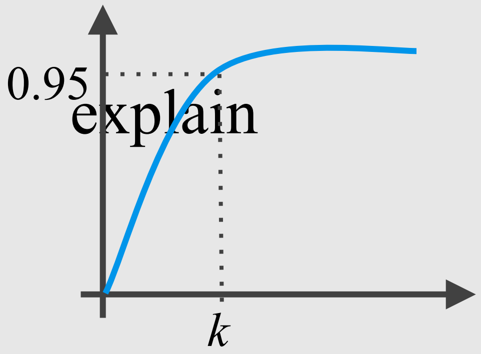
How many PCs?

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

How many PCs?

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

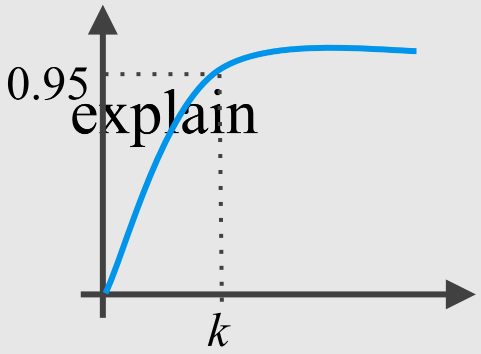
$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \leq 1$$



How many PCs?

- Have eigenvectors $u_1, u_2, \ldots, u_n$, want $k < n$
- eigenvalue λ_i = variance along u_i
- Pick u_i that explain the most variance:
  - Sort eigenvectors s.t. $\lambda_1 > \lambda_2 > \lambda_3 > \ldots > \lambda_n$
  - 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 $\mathbf{\Sigma}$
3. Find eigenvectors u and eigenvalues λ
4. Sort eigenvectors and pick first k eigenvectors
5. Project data to k eigenvectors

PCA Algorithm

By Singular Value Decomposition

Data Preprocessing

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

Preprocessing (feature scaling/mean normalization):

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

Center the data

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":

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

Compute "eigenvectors" of matrix **Σ**:

[U, S, V] = svd(sigma) ➡ Singular Value Decomposition

PCA Algorithm

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

Compute "covariance matrix":

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

Compute "eigenvectors" of matrix **Σ**:

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

PCA Algorithm

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

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

PCA Algorithm

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

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

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

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

PCA Algorithm

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

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

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

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

PCA Algorithm

After mean normalization and optionally feature scaling:

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

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

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

Choosing the Number of Principal Components

Choosing *k* (#Principal Components)

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

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

Choosing k (#Principal Components)

$$x_{\text{approx}} = (\mathrm{U}_{\text{reduce}})\, z^{(i)}$$

$$n \times k \qquad n \times 1$$

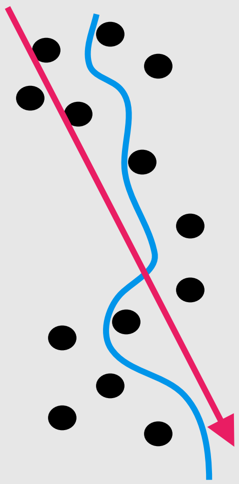
[U, S, V] = svd(sigma)

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

Practical Issues

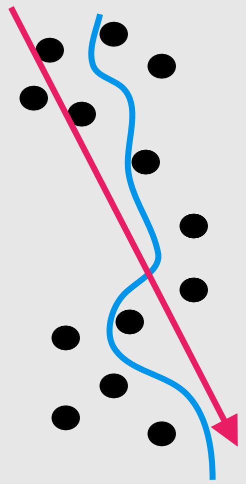
PCA: Practical Issues

- PCA assumes underlying subspace is linear
  - PCA cannot find a curve



PCA: Practical Issues

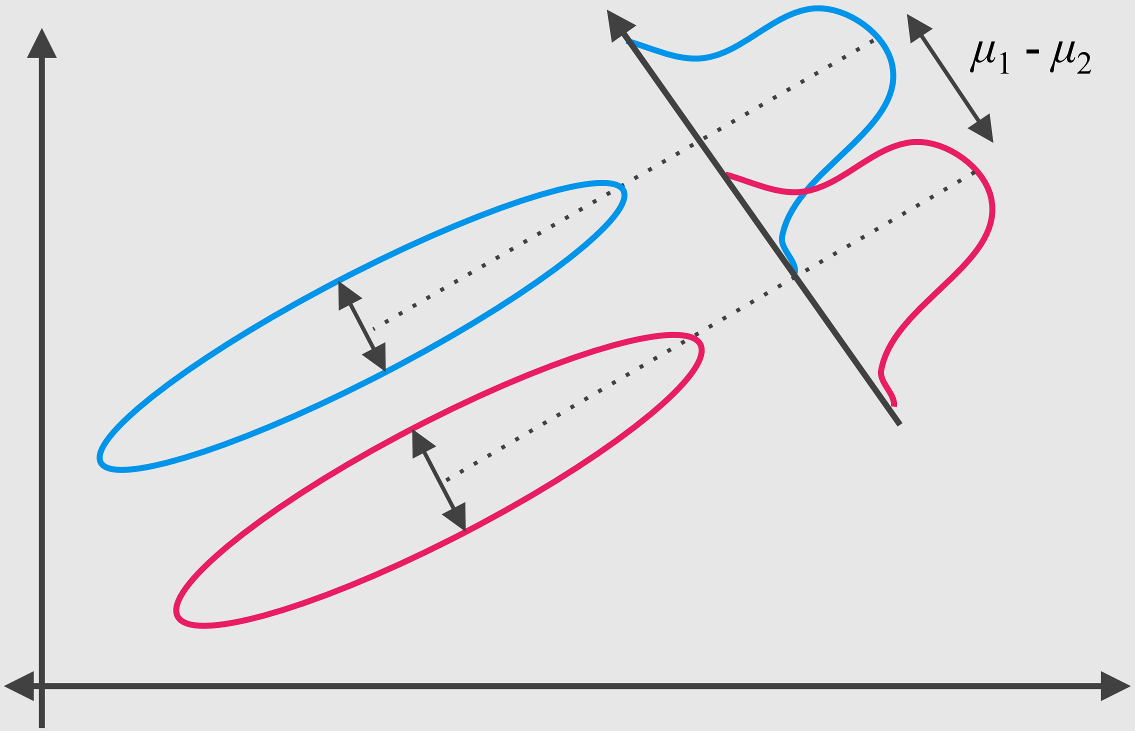
- PCA assumes underlying subspace is linear
  - PCA cannot find a curve
- PCA and Classification
  - PCA is unsupervised
  - PCA can pick direction that makes hard to separate classes



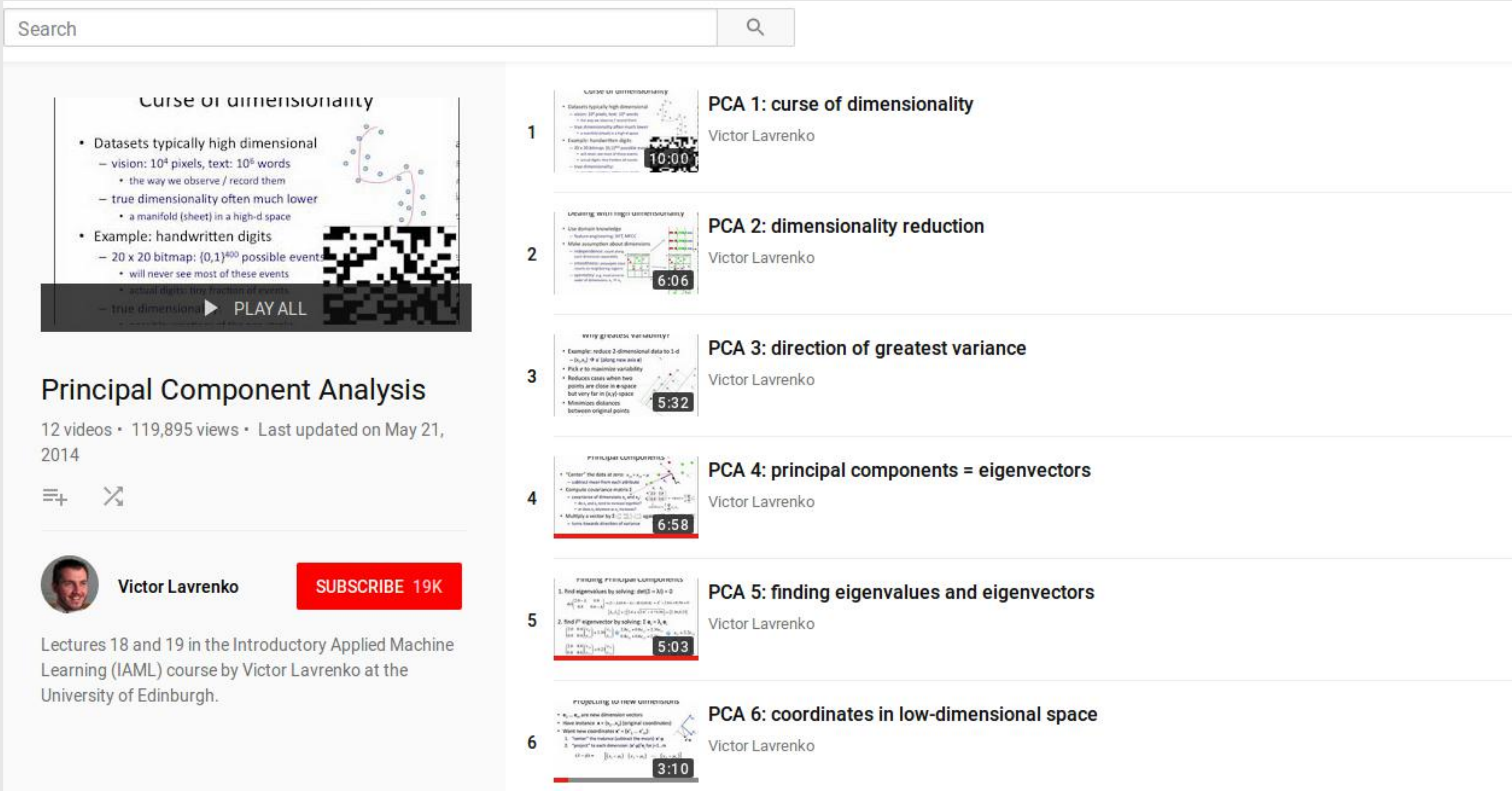
Linear Discriminant Analysis (LDA)

- LDA pick a new dimension that gives:
  - Maximum separation between means of projected classes
  - Minimum variance within each projected class
- Solution: eigenvectors based on between-class and within-class covariance matrix

Linear Discriminant Analysis (LDA)



https://www.youtube.com/playlist?list=PLBv09BD7ez_5_yapAg86Od6JeeypkS4YM



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