Recall from last time ...

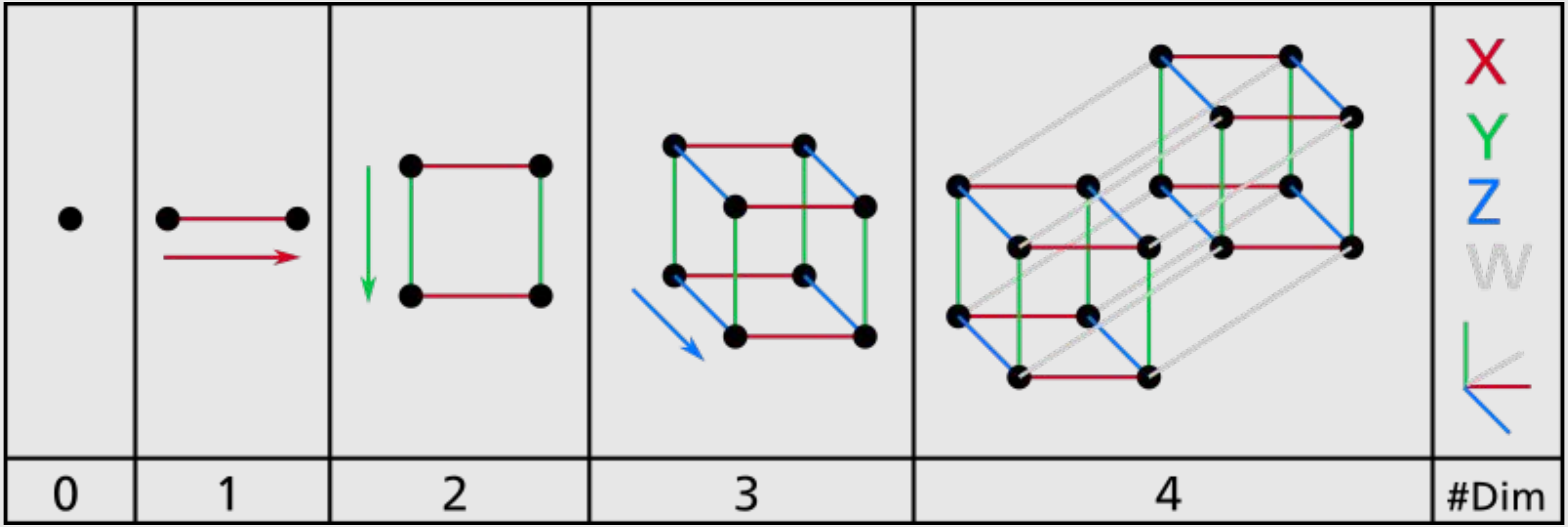
Why is Dimensionality Reduction useful?

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
- Data Visualization
- To mitigate **“the curse of dimensionality”**

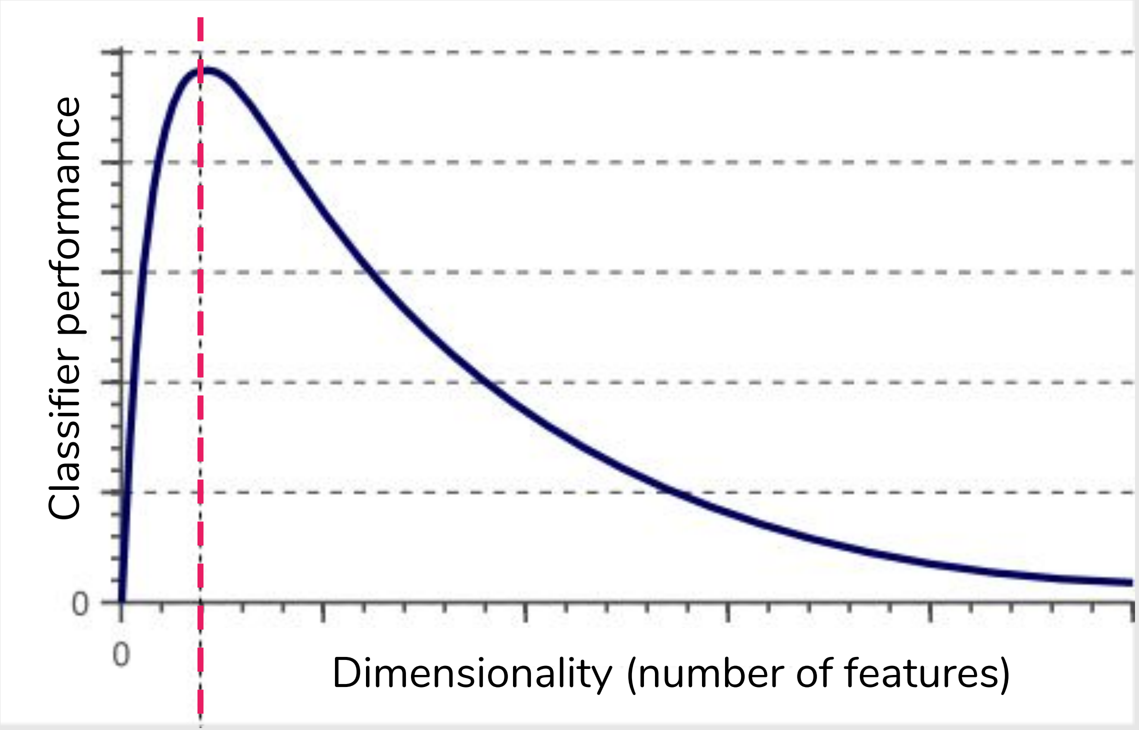
The Curse of Dimensionality

The Curse of Dimensionality



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

The Curse of Dimensionality



Optimal number of features

How to reduce dimensionality?

How to reduce dimensionality?

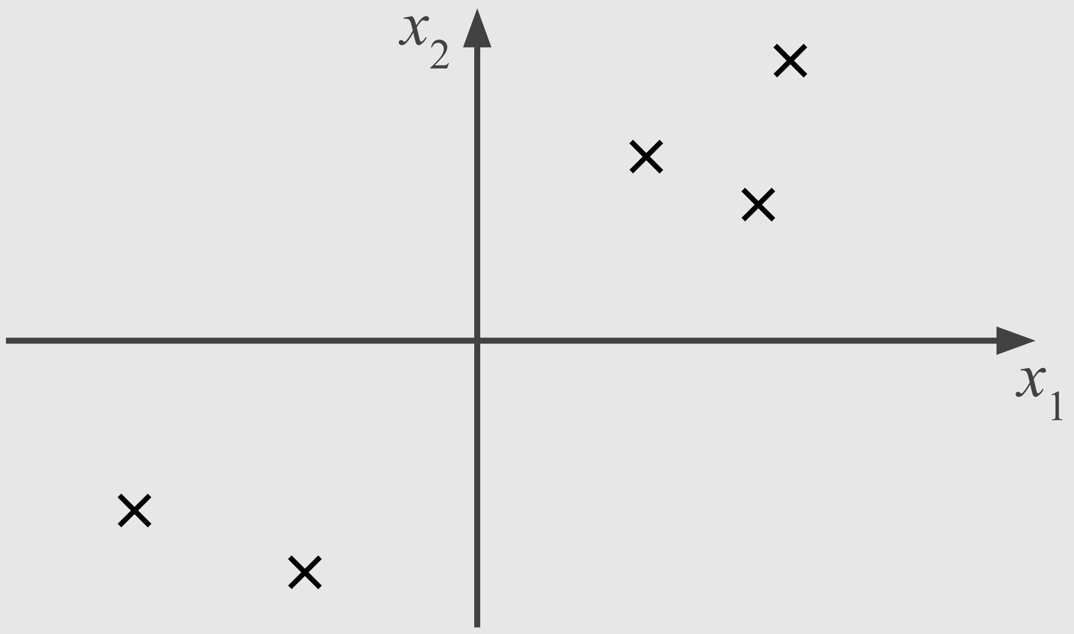
- **Feature Selection:** choosing a subset of all the features (the ones more informative).
  - $\mathbf{x_1}, x_2, \mathbf{x_3}, x_4, \mathbf{x_5}$
- **Feature Extraction:** create a subset of new features by combining the existing ones.
  - $z = f(x_1, x_2, x_3, x_4, x_5)$

PCA: Principal Component Analysis

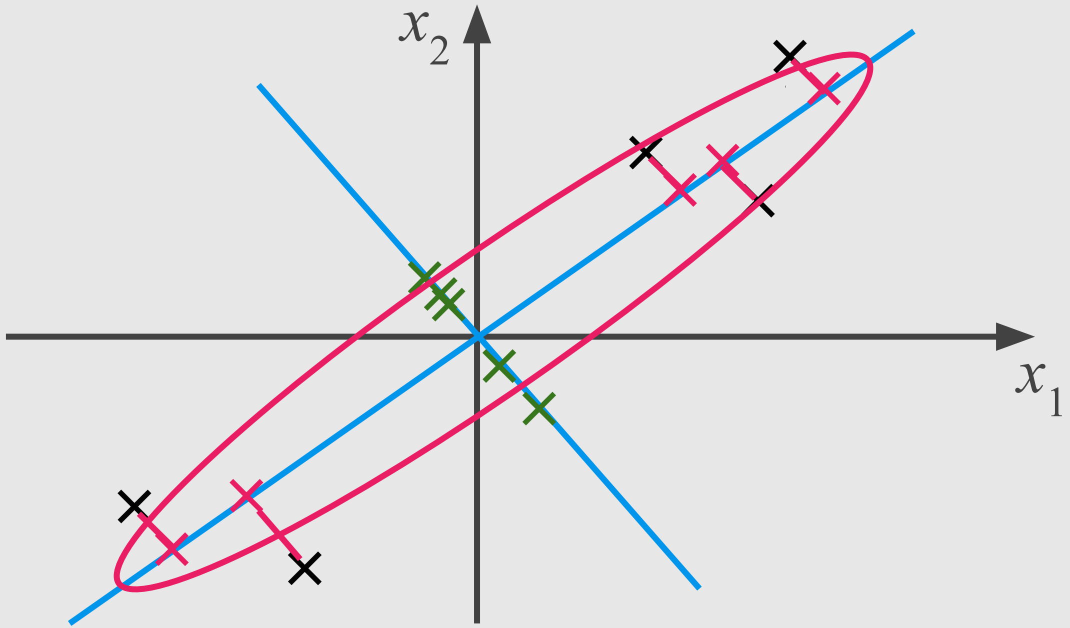
Principal Component Analysis (PCA)

- 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.

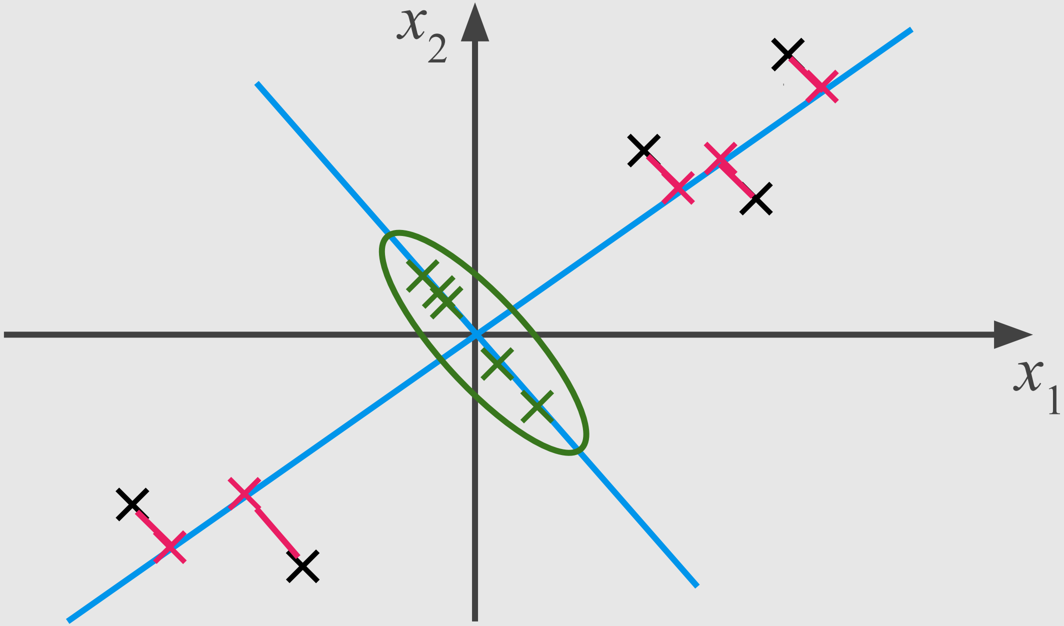
Problem Formulation (PCA)



Problem Formulation (PCA)

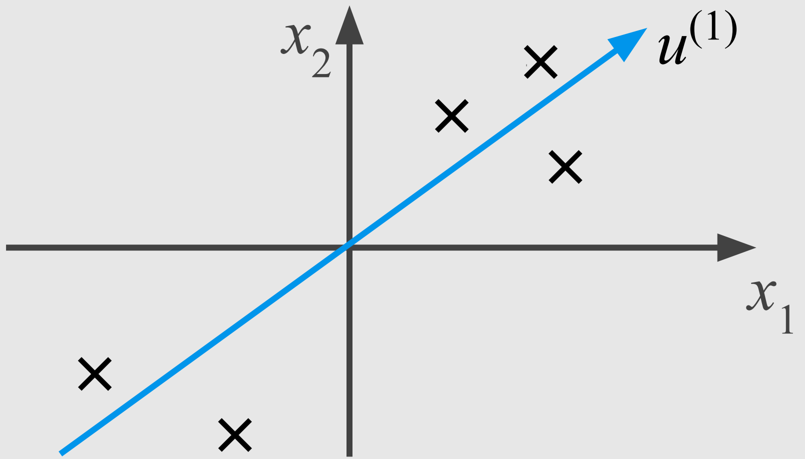


Problem Formulation (PCA)



Problem Formulation (PCA)

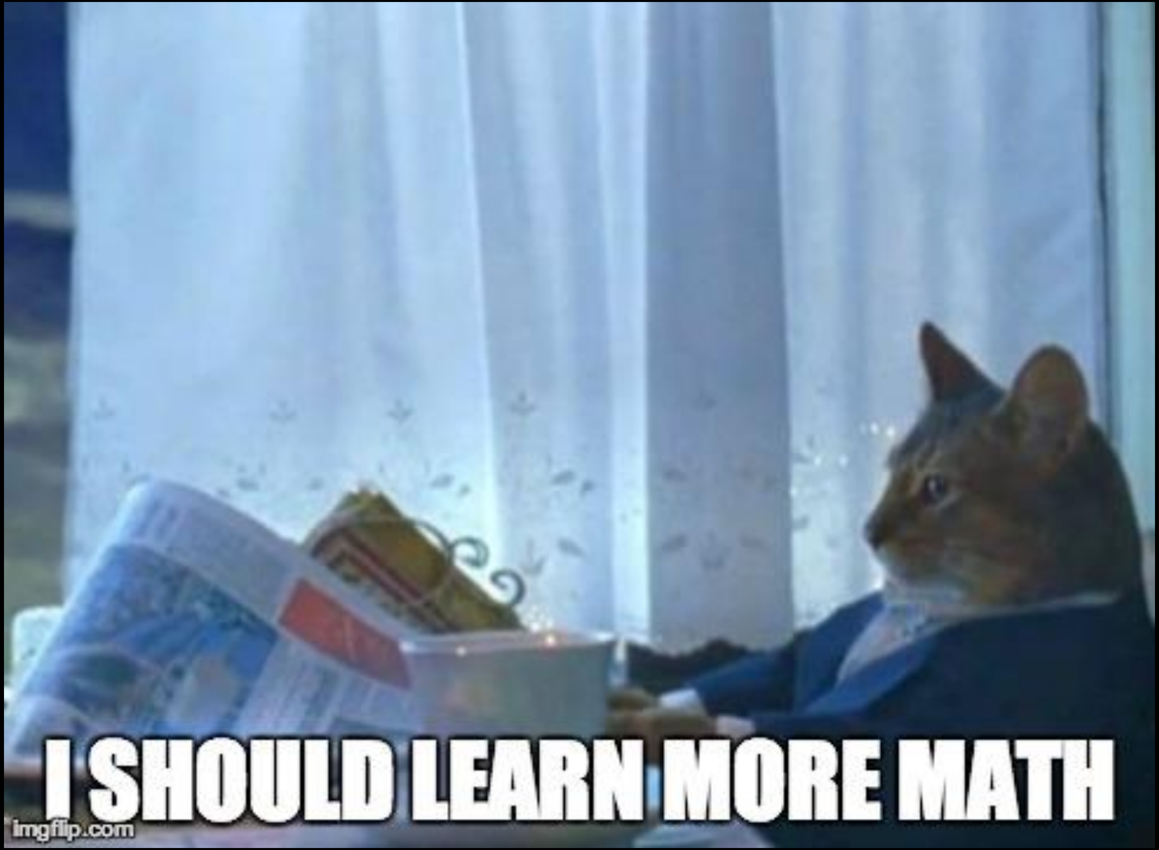
- Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.



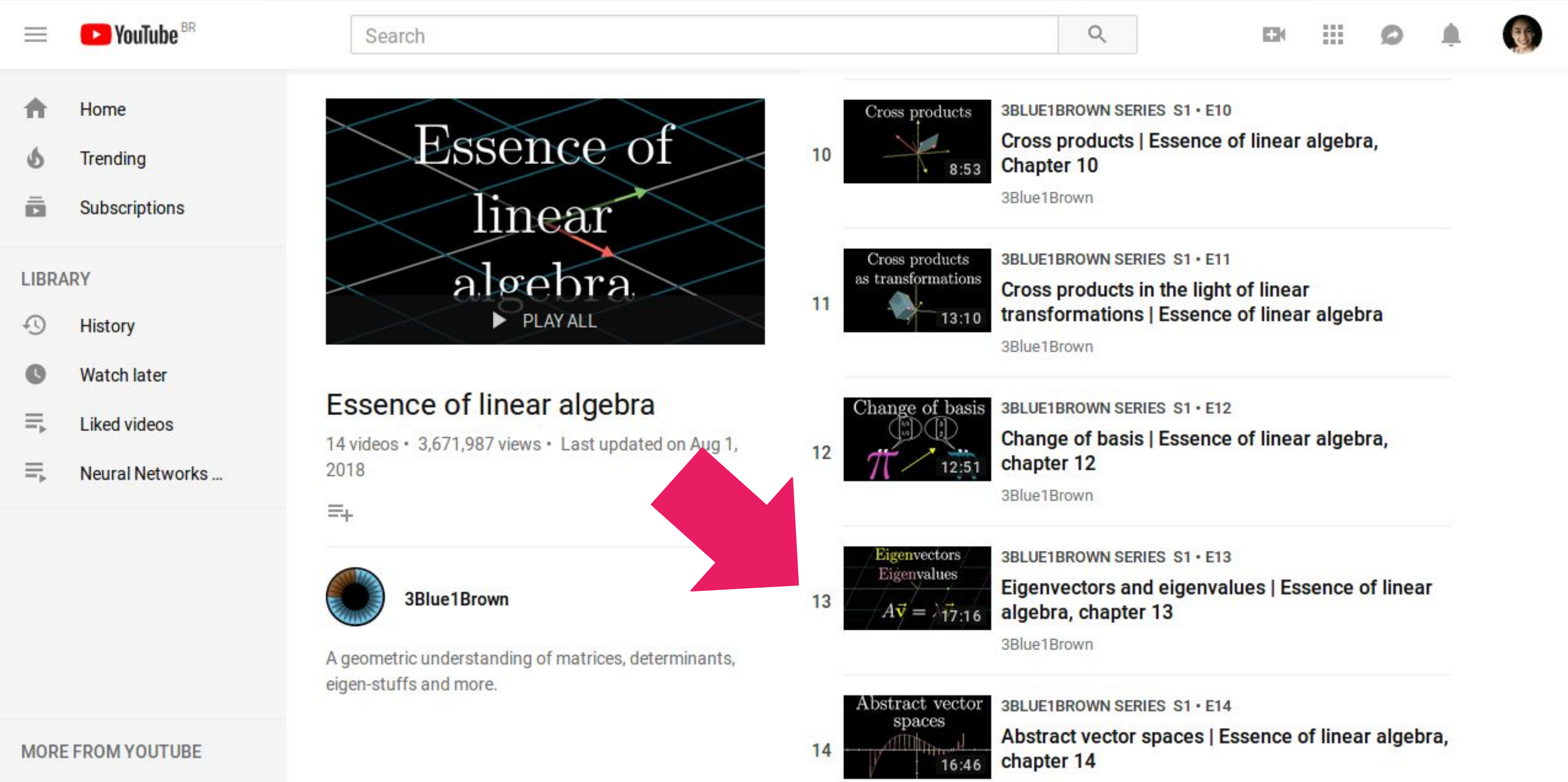
Problem Formulation (PCA)

- 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.

PCA Algorithm
By Singular Value Decomposition


I SHOULD LEARN MORE MATH
imgflip.com

https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab



Data Preprocessing

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

Preprocessing (feature scaling/mean normalization):

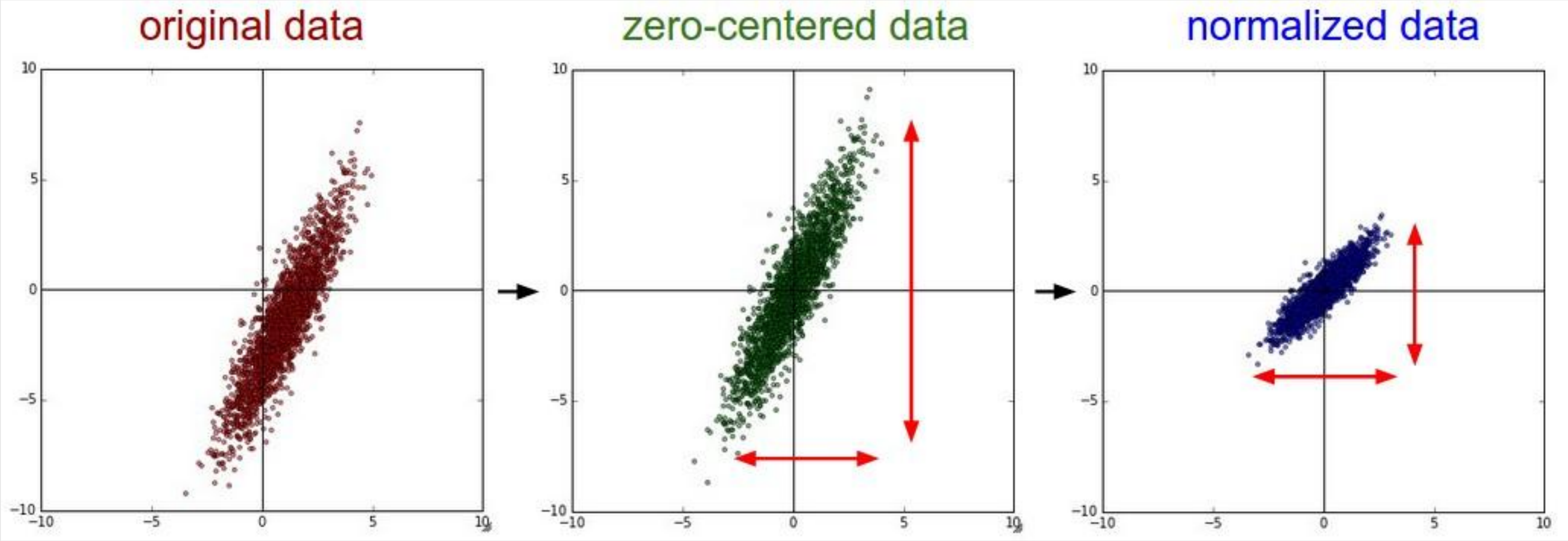
$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

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":

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

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

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

PCA Algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

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

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

eigenvalues

PCA Algorithm

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

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

PCA Algorithm

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

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

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

PCA Algorithm

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

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

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

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

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

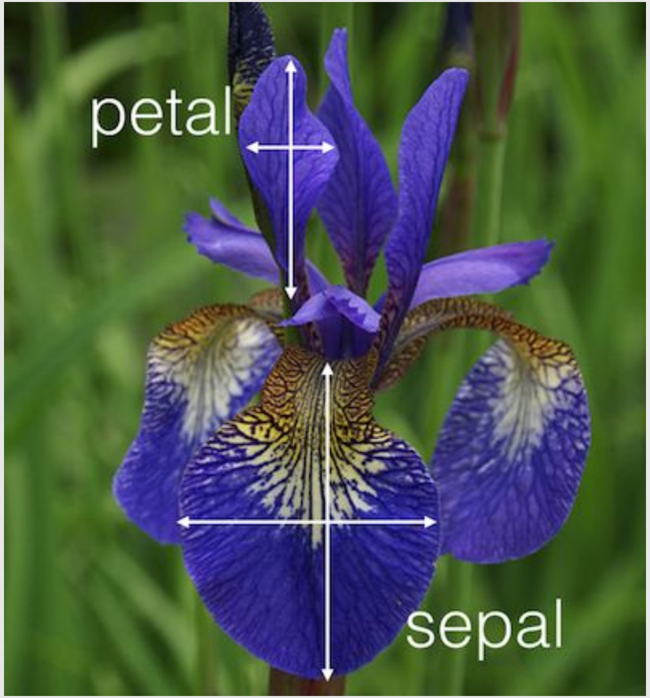
Choosing k (#Principal Components)

eigenvalues

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

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

Using PCA (Iris Dataset)



150 iris flowers from three different species.

The three classes in the Iris dataset:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

The four features of the Iris dataset:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

PCA Algorithm
By Eigen Decomposition

PCA in a Nutshell (Eigen Decomposition)

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

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 in a Nutshell (Eigen Decomposition)

1. Center the data (and normalize)
2. **Compute covariance matrix $\boldsymbol{\Sigma}$**
3. Find eigenvectors u and eigenvalues λ
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":

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

PCA Algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

Covariance of dimensions x_1 and x_2:

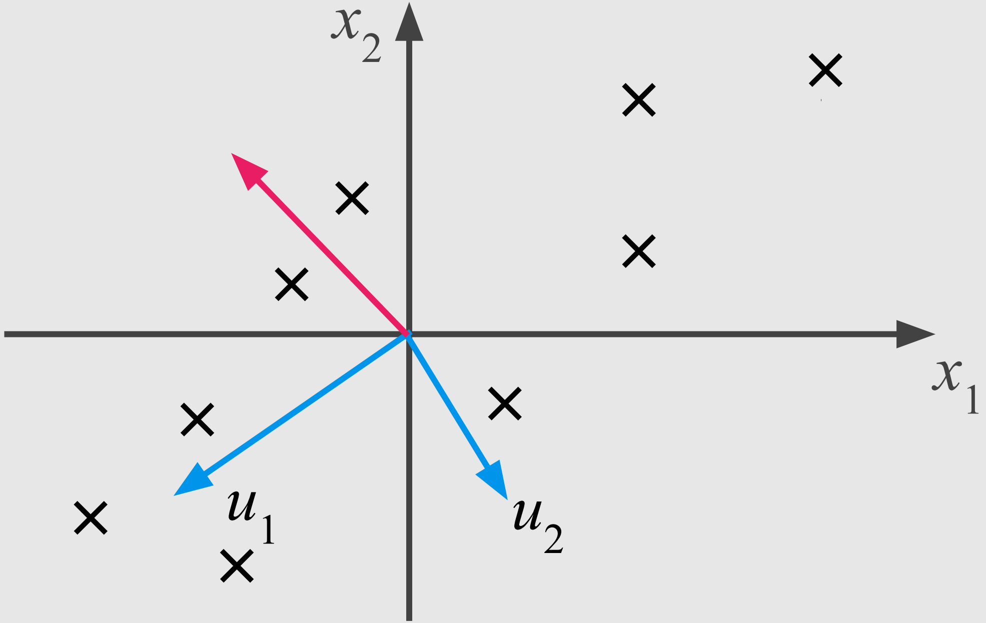
- Do x_1 and x_2 tend to increase together?
- or does x_2 decrease as x_1 increases?

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

PCA Algorithm

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

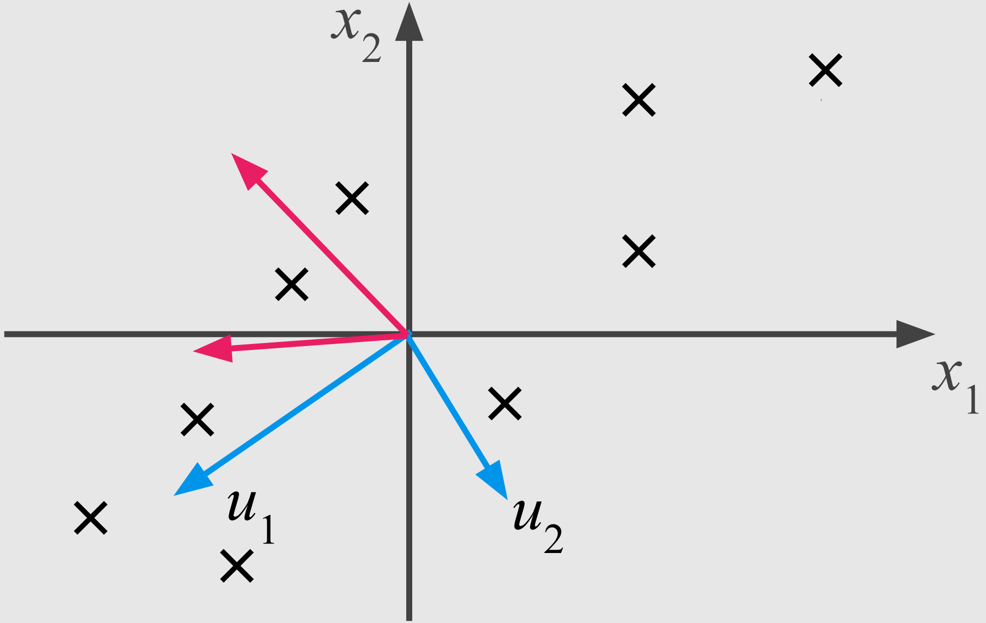
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$



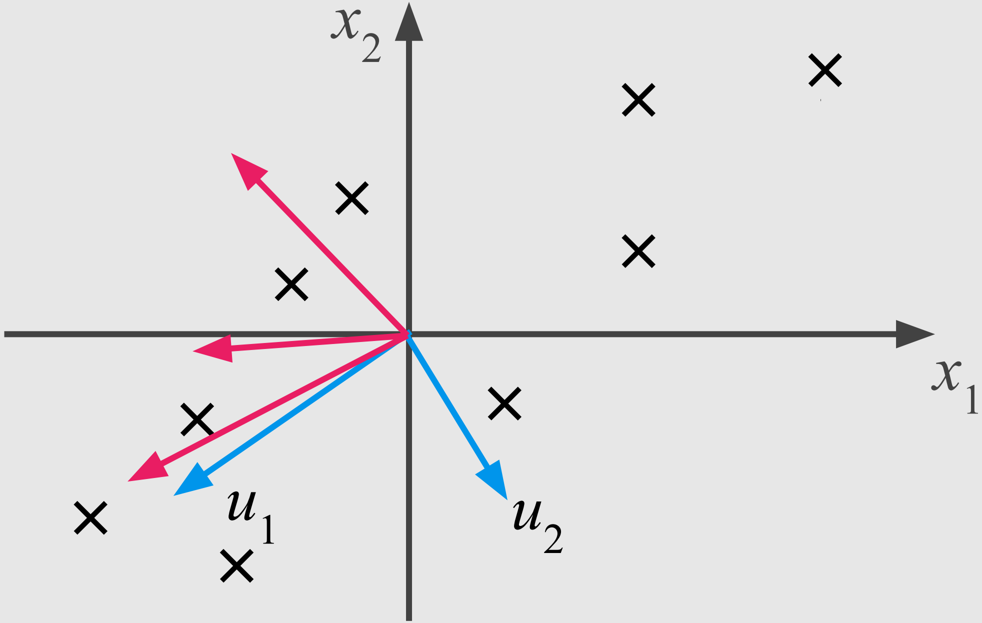
PCA Algorithm

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

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



PCA Algorithm

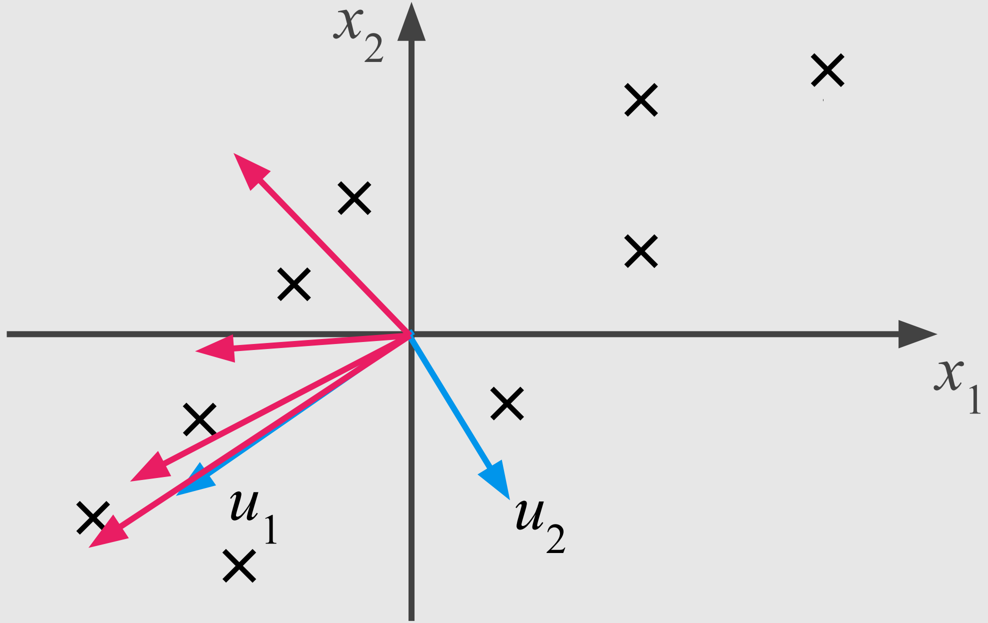


Multiple a vector by $\boldsymbol{\Sigma}$:

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

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

PCA Algorithm



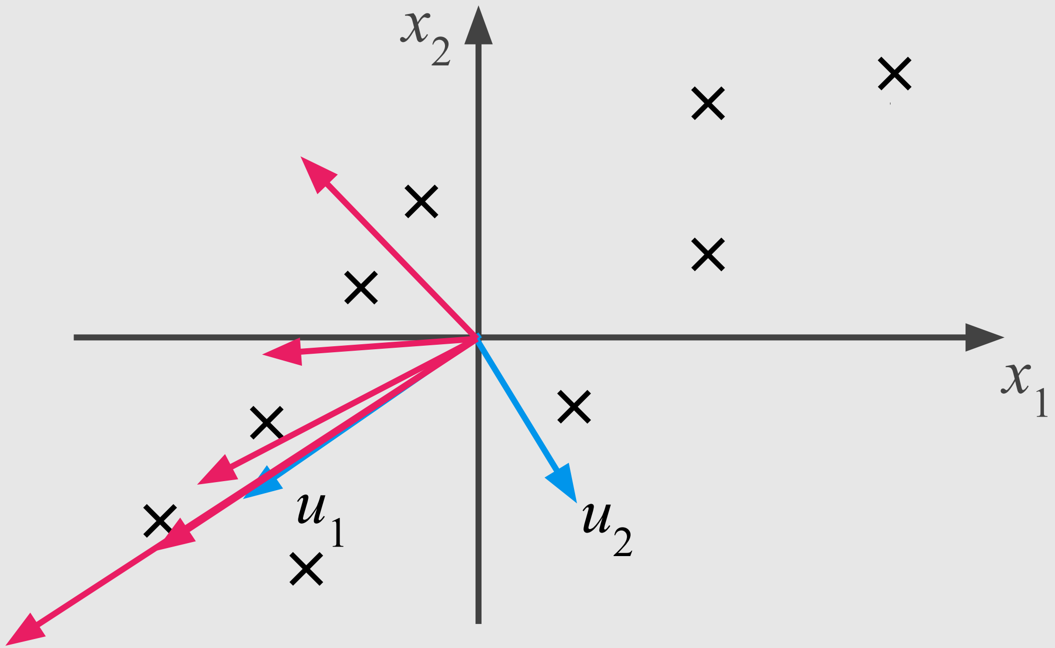
Multiple a vector by $\boldsymbol{\Sigma}$:

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

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

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

PCA Algorithm



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

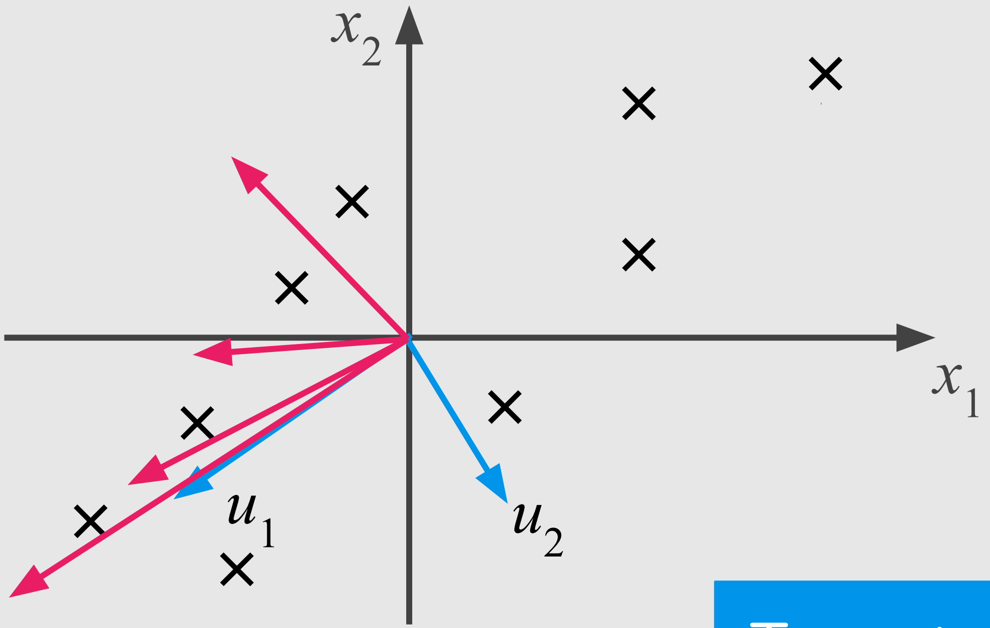
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix}$$

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

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

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

PCA Algorithm



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

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

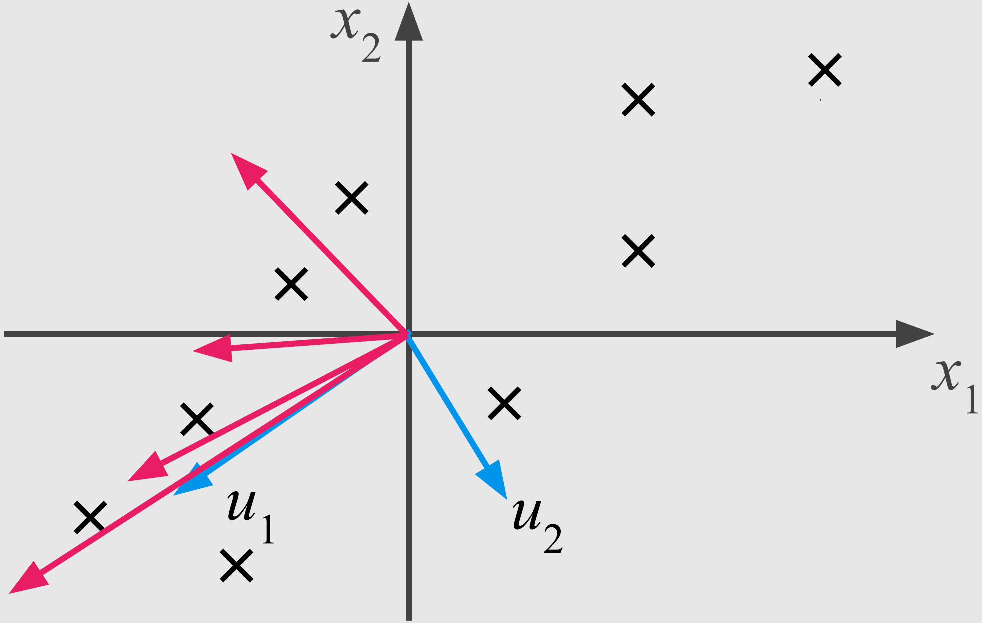
$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \times \begin{bmatrix} -1.2 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -1.0 \end{bmatrix}$$

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

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

Turns towards direction of variation

PCA Algorithm

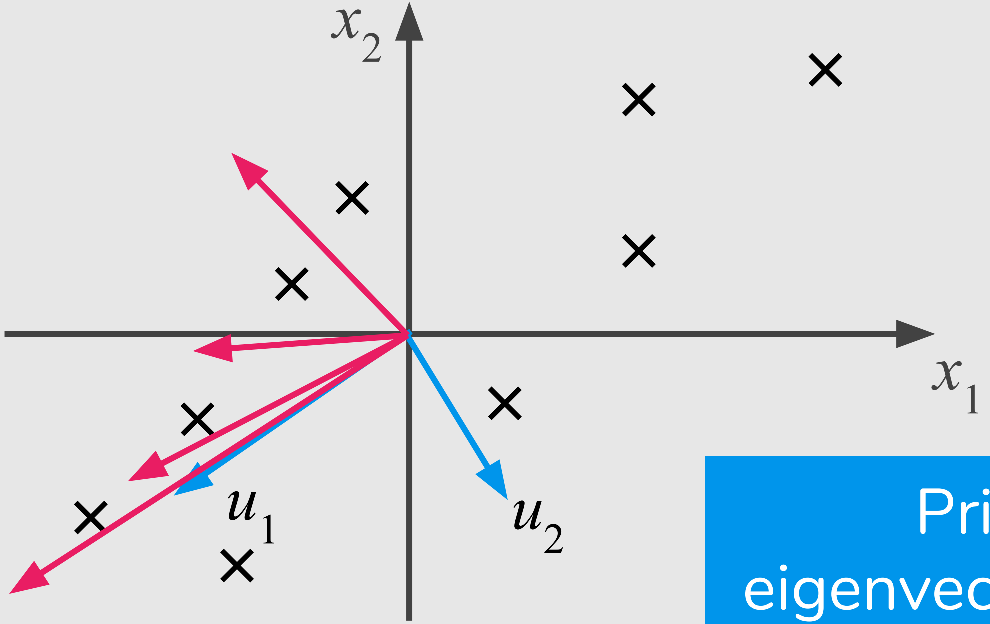


Want vectors u which aren't turned: $\mathbf{\Sigma} u = \lambda u$

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

λ = 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

PCA in a Nutshell (Eigen Decomposition)

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

Finding Principal Components

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

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

Finding Principal Components

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

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

Finding Principal Components

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

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

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

Finding Principal Components

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

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: $\mathbf{\Sigma} u_i = \lambda_i u_i$

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

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: $\mathbf{\Sigma} u_i = \lambda_i u_i$

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

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

$$u_{11} = 2.2u_{12}$$

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

Want $\|u_1\|=1$

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

Finding Principal Components

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

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

$$\Downarrow$$

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

$\Downarrow$ 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: $\mathbf{\Sigma} u_i = \lambda_i u_i$

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

$$\Downarrow$$

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

$$\Downarrow \text{ Want } \|u_1\|=1$$

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

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

3. 1^{st} PC: $\begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}$ and 2^{nd} PC: $\begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix}$

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 eigenvalues and pick first k eigenvectors**
5. Project data to k eigenvectors

How many PCs?

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

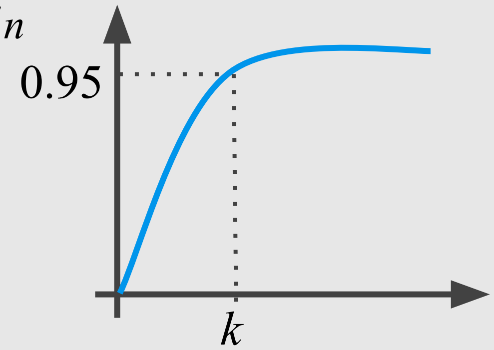
How many PCs?

- 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

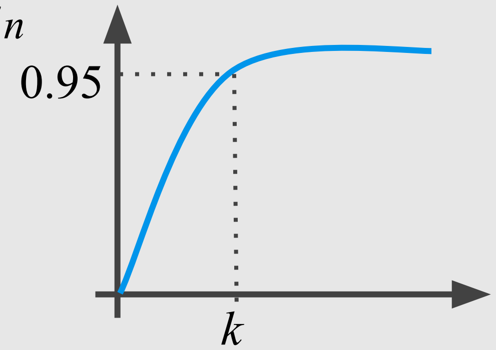
$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \leq 1$$



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%

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

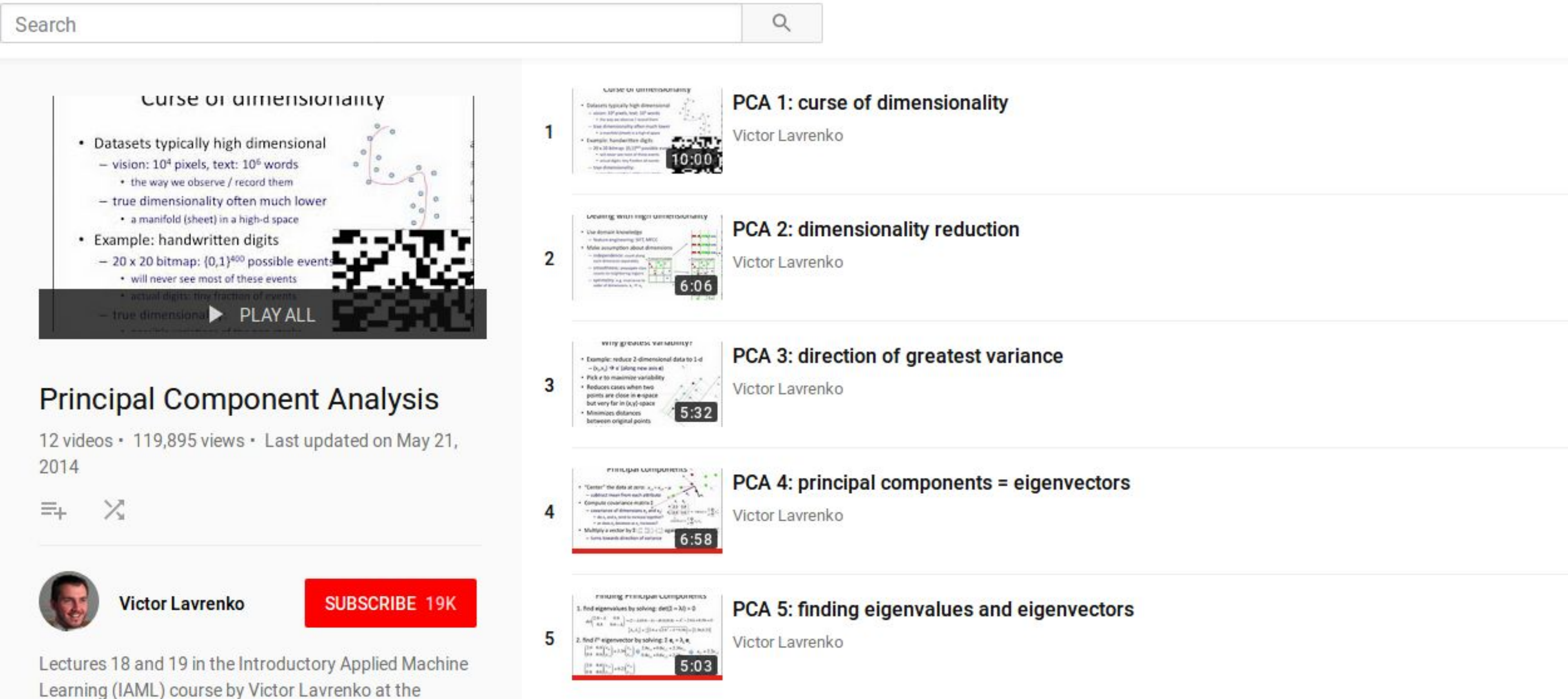


PCA in a Nutshell (Eigen Decomposition)

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

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

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

Linear Discriminant Analysis

Machine Learning and Pattern Recognition

Prof. Sandra Avila

Institute of Computing (IC/Unicamp)

MC886/MO444, September 27, 2018

Today's Agenda

- Linear Discriminant Analysis
  - PCA vs LDA
  - LDA: Simple Example
  - LDA Algorithm
  - LDA Step by Step

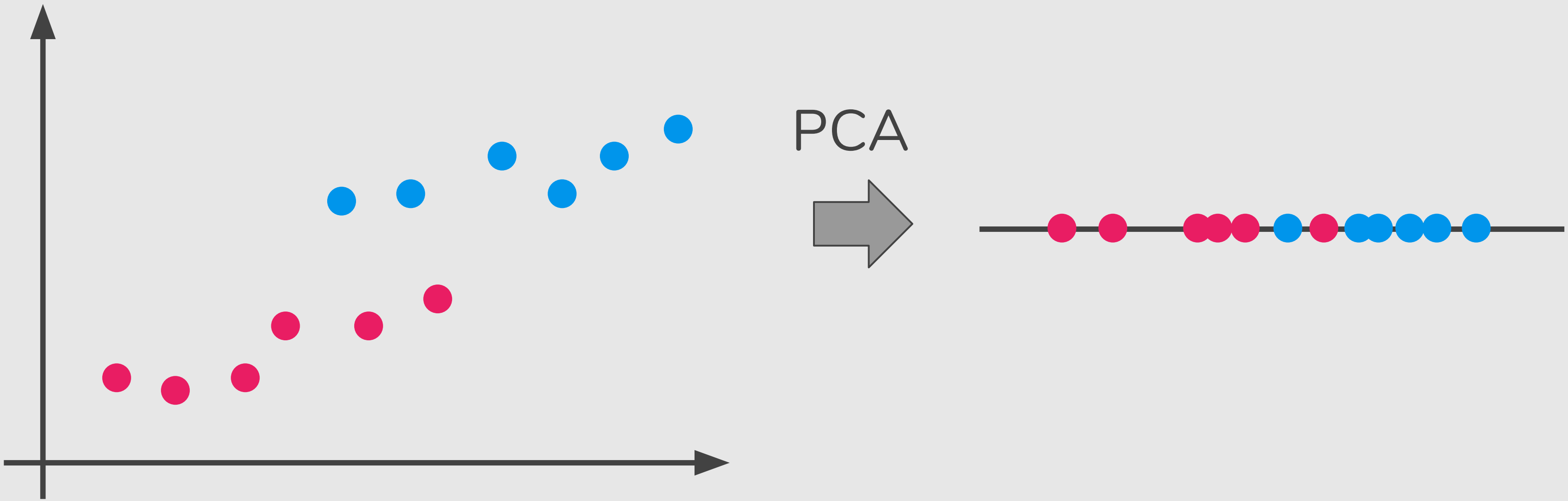
Linear Discriminant Analysis

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

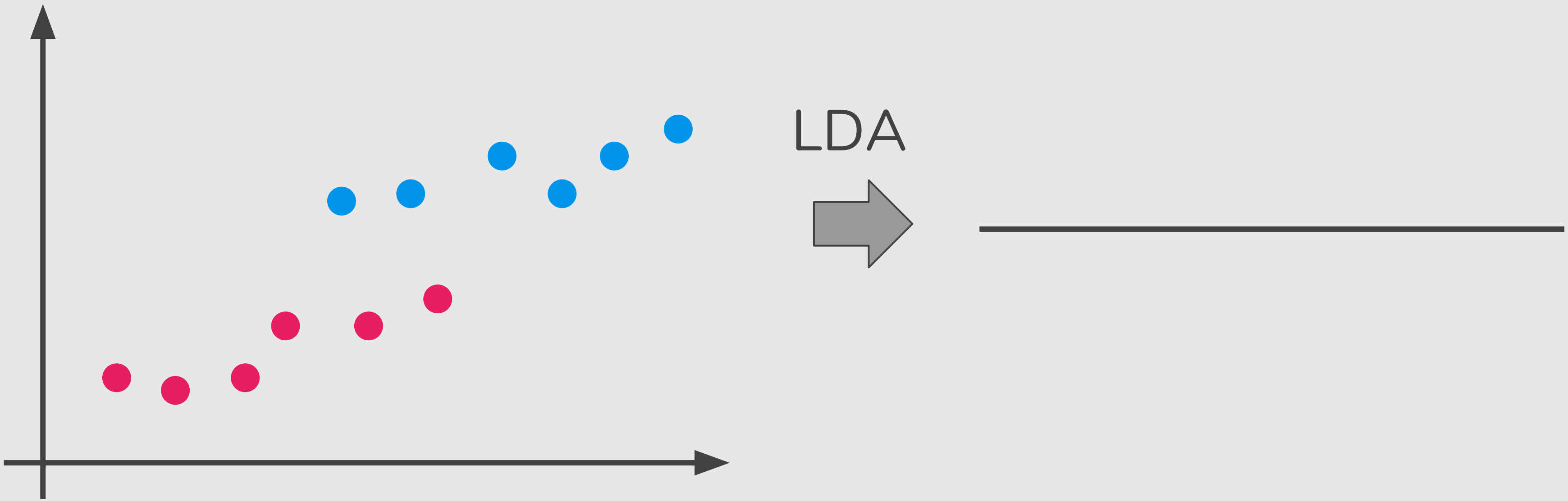
LDA: Simple Example

Reducing 2D to 1D



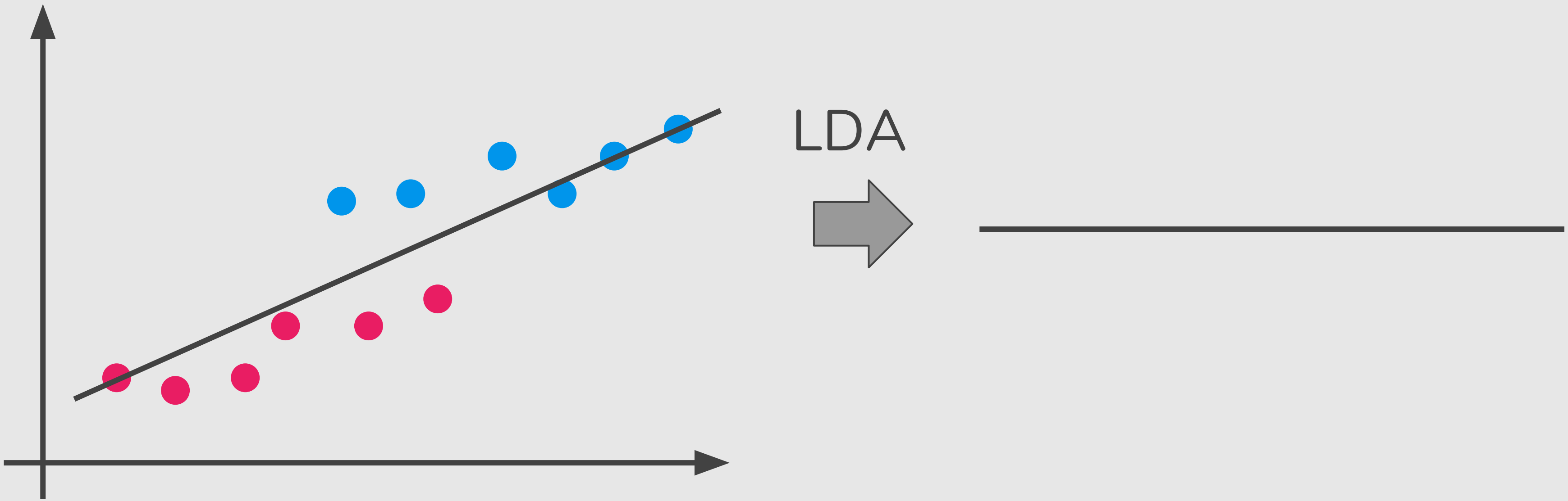
LDA: Simple Example

Reducing 2D to 1D



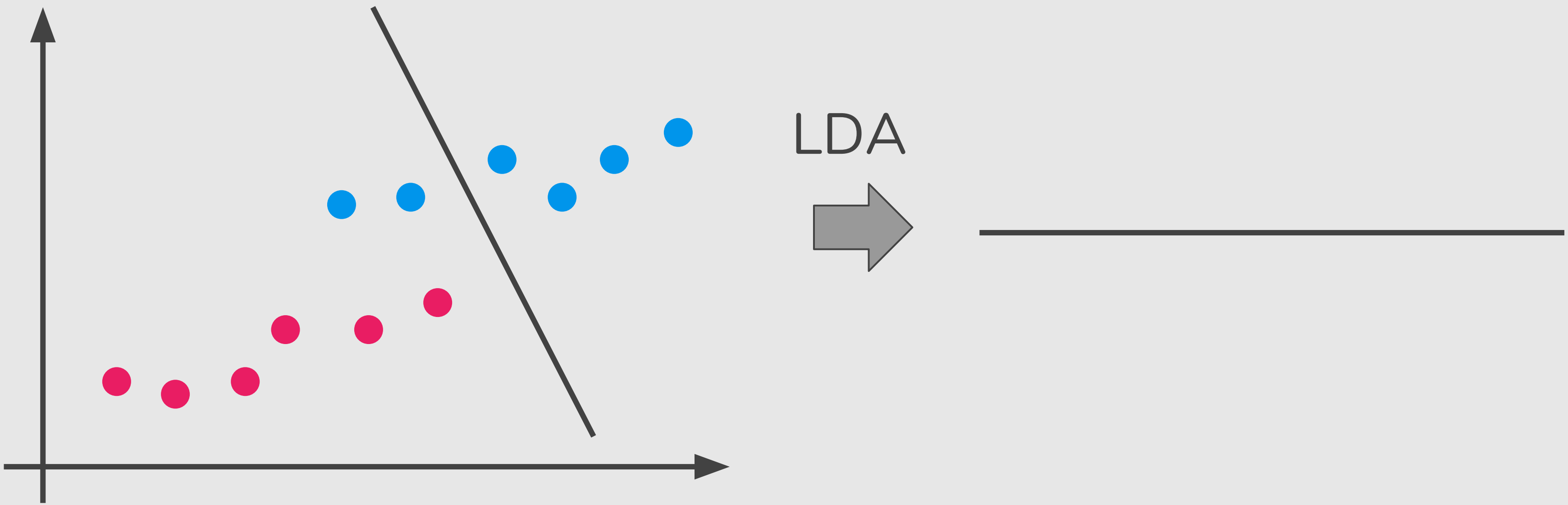
LDA: Simple Example

Reducing 2D to 1D



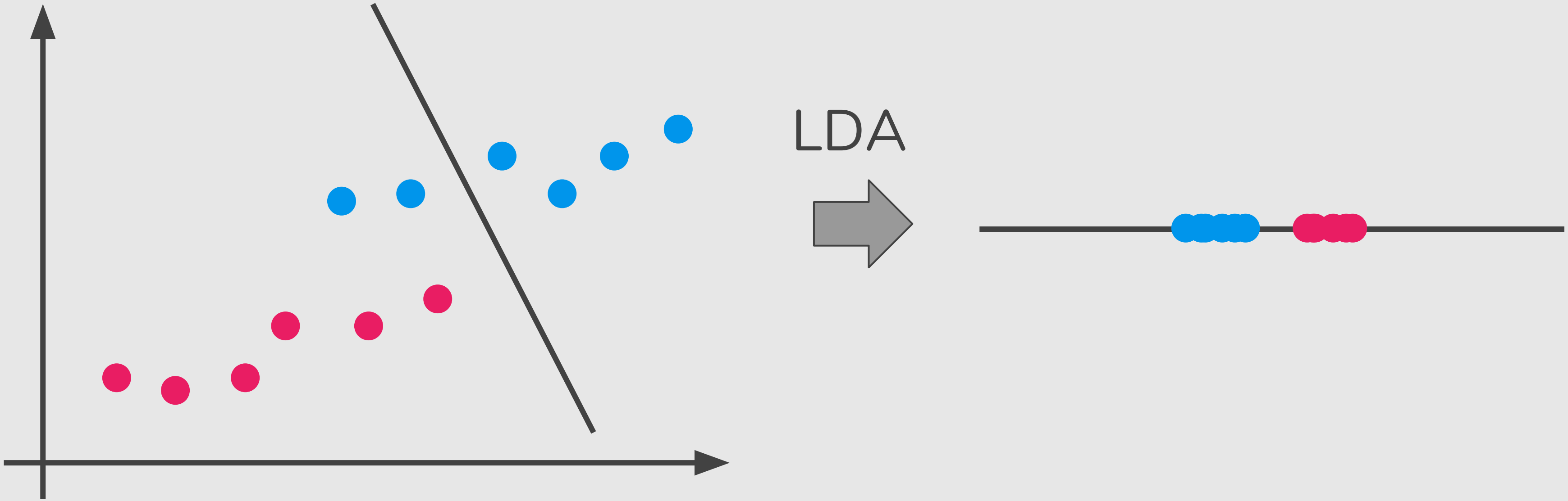
LDA: Simple Example

Reducing 2D to 1D



LDA: Simple Example

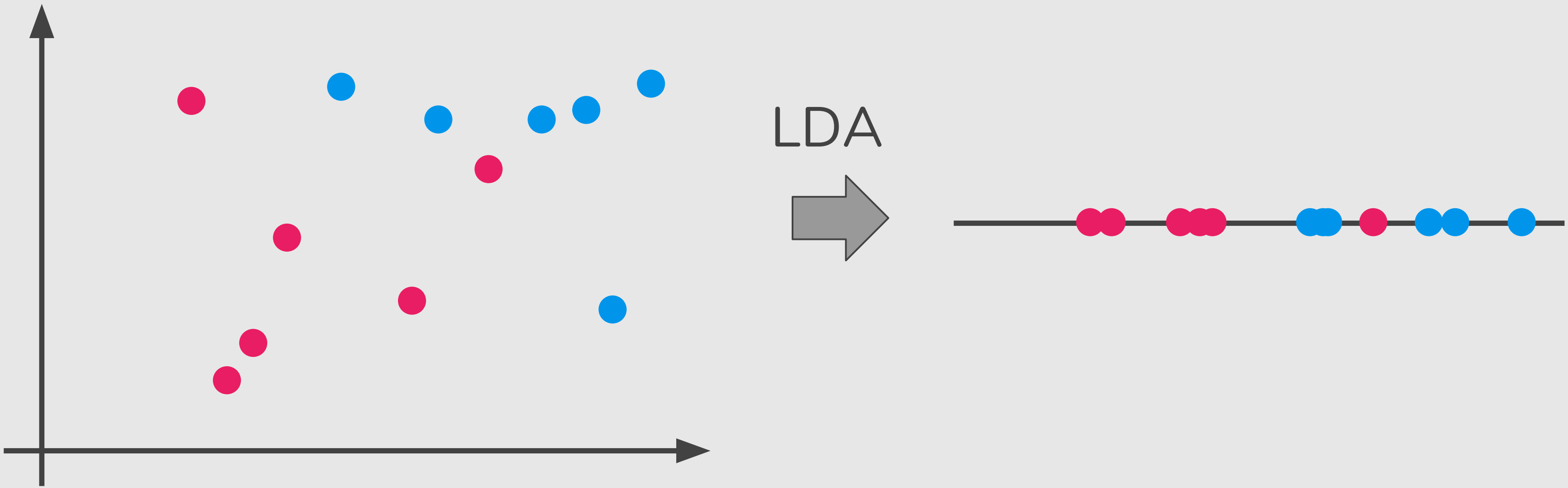
Reducing 2D to 1D



How LDA create a new axis?

How LDA create a new axis?

Reducing 2D to 1D



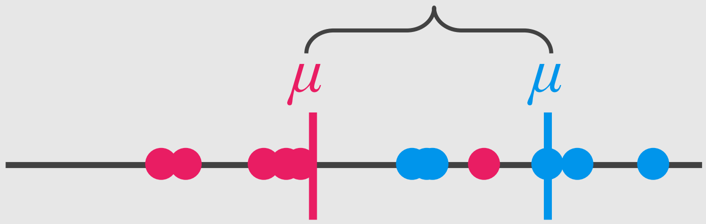
How LDA create a new axis?

The new axis is created according two criteria:

How LDA create a new axis?

The new axis is created according two criteria:

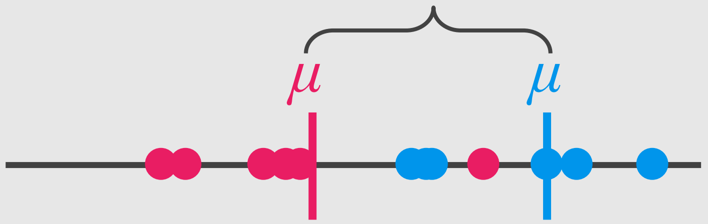
1. Maximize the distance between the means:



How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:

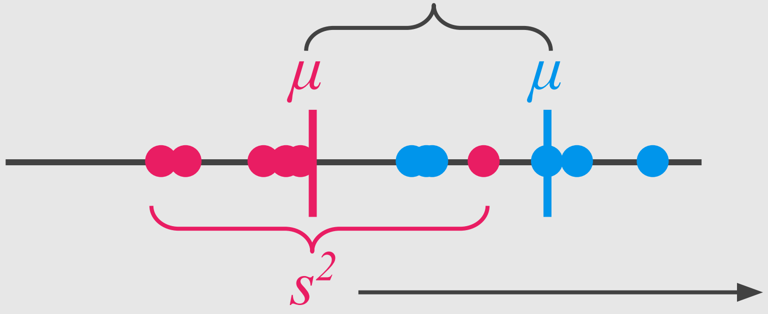


2. Minimize the variation (which LDA calls scatter) within each class.

How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:



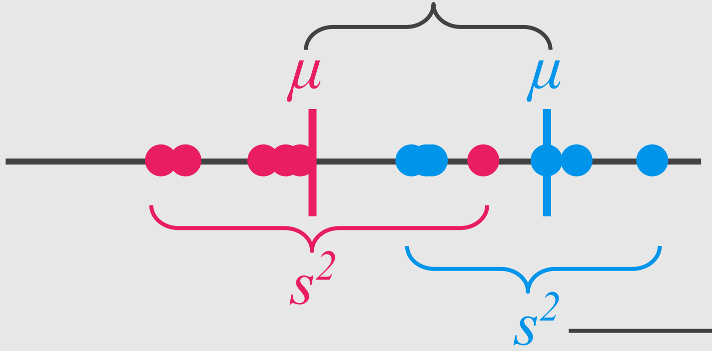
This is the scatter around the pink dots.

2. Minimize the variation (which LDA calls scatter) within each class.

How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:



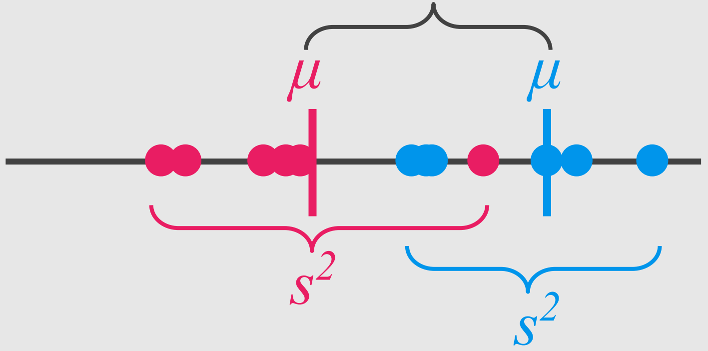
This is the scatter around the blue dots.

2. Minimize the variation (which LDA calls scatter) within each class.

How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:



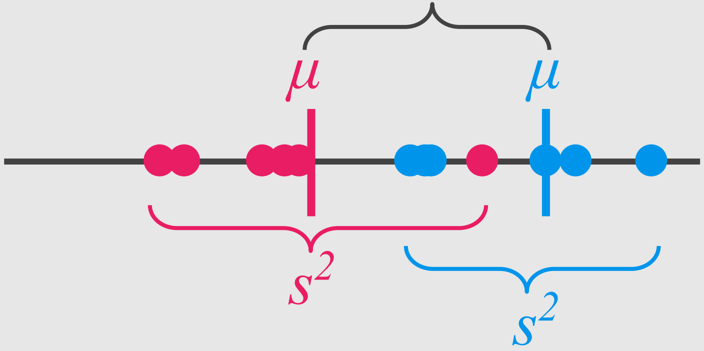
$$\frac{(\mu - \mu)^2}{s^2 + s^2}$$

2. Minimize the variation (which LDA calls scatter) within each class.

How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:



2. Minimize the variation (which LDA calls scatter) within each class.

$$\frac{(\mu - \mu)^2}{s^2 + s^2}$$

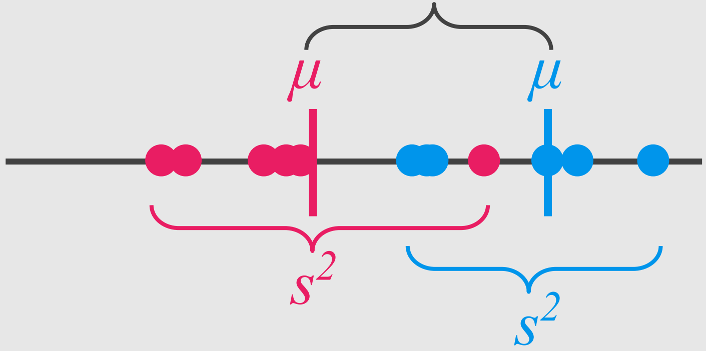
→ Ideally large

→ Ideally small

How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:



2. Minimize the variation (which LDA calls scatter) within each class.

Let's call $(\mu - \mu)$ d for distance.

$$\frac{(\mu - \mu)^2}{s^2 + s^2}$$

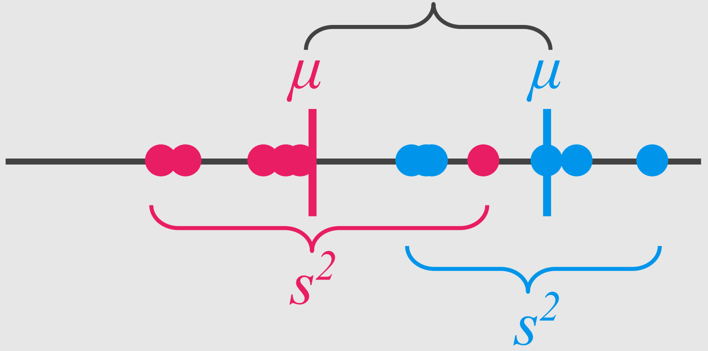
$(\mu - \mu)^2 \longrightarrow$ Ideally large

$s^2 + s^2 \longrightarrow$ Ideally small

How LDA create a new axis?

The new axis is created according two criteria:

1. Maximize the distance between the means:



2. Minimize the variation (which LDA calls scatter) within each class.

Let's call $(\mu - \mu)$ d for distance.

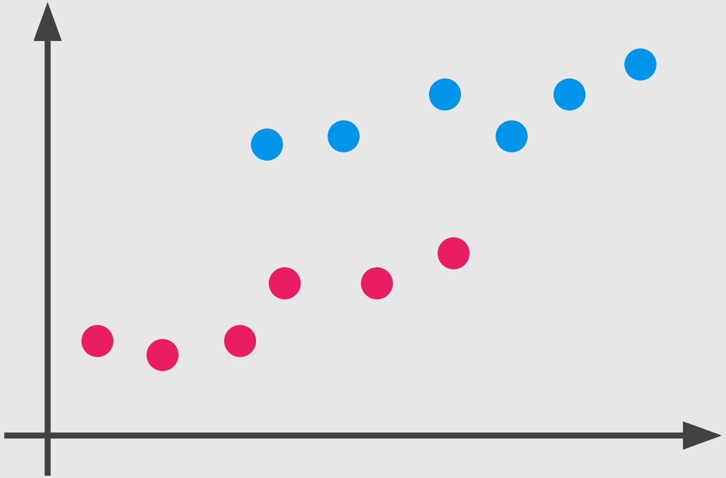
$$\frac{d^2}{s^2 + s^2}$$

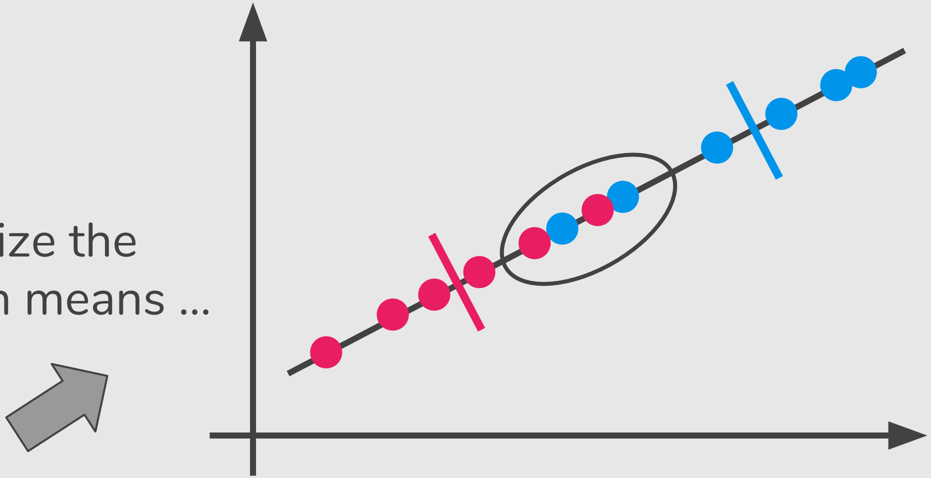
$\longrightarrow$ Ideally large

$\longrightarrow$ Ideally small

Why both distance and scatter are important?

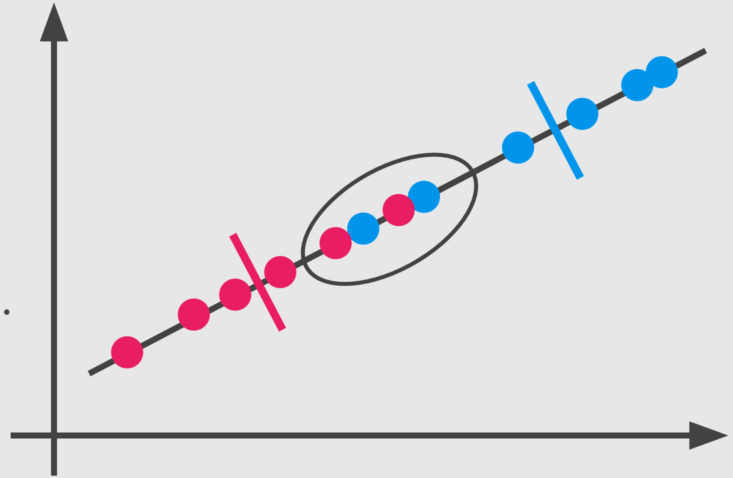
If we only maximize the distance between means ...

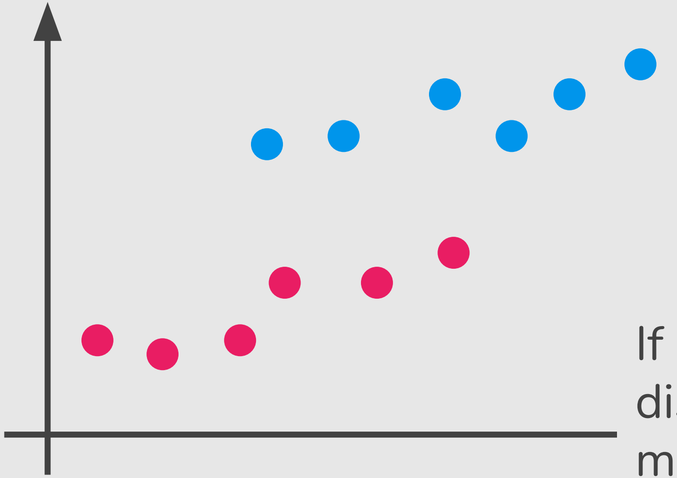


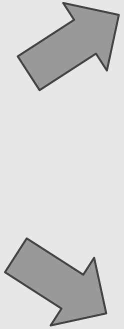


Why both distance and scatter are important?

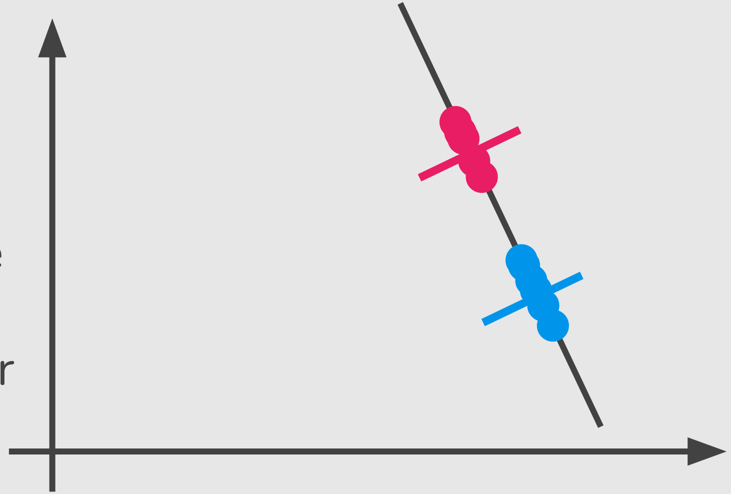
If we only maximize the distance between means ...



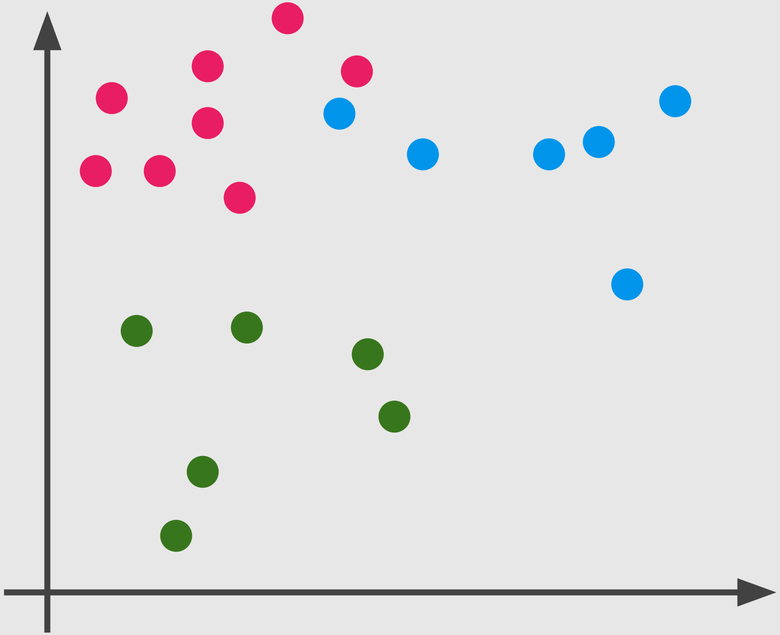




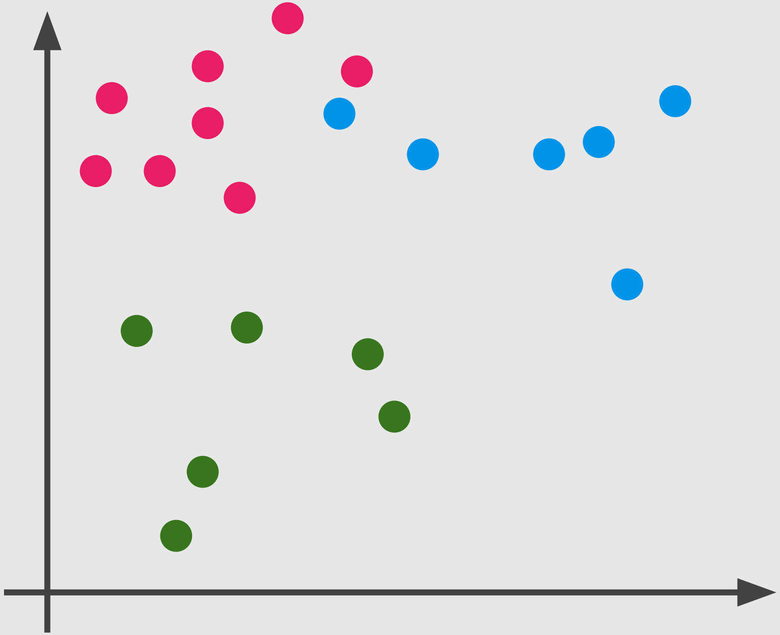
If we optimize the distance between means and scatter



What if we have 3 classes?



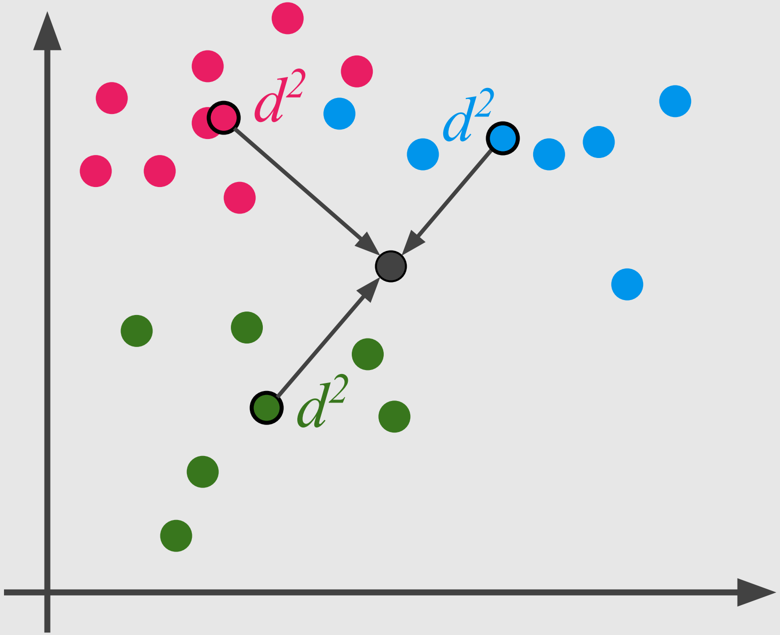
What if we have 3 classes?



How we measure the distance among the means?

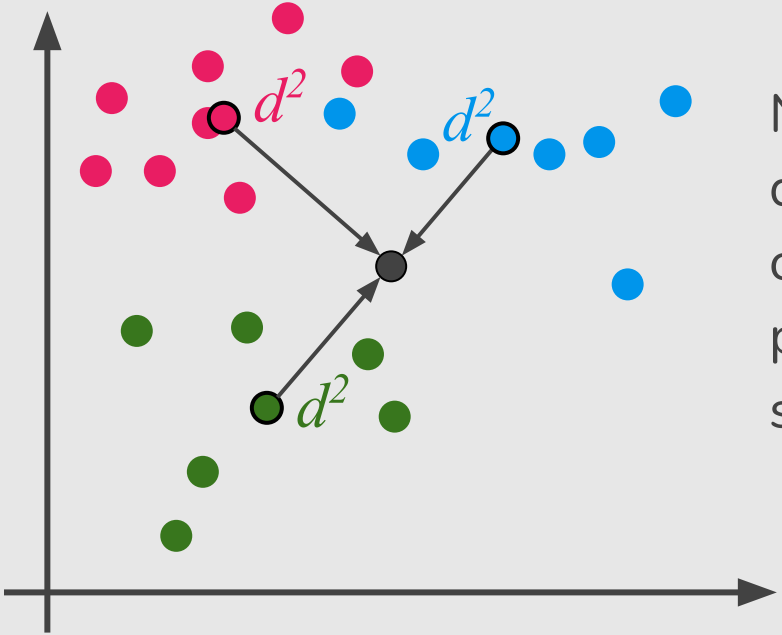
What if we have 3 classes?

What if we have 3 classes?



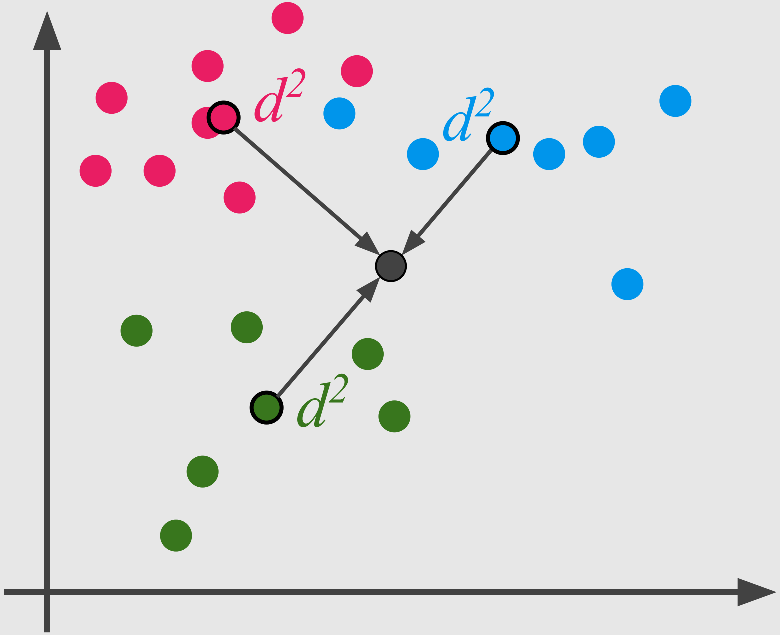
Then measure the distance between a point that is central in each class and the main central point.

What if we have 3 classes?



Now maximize the distance between each class and the central point while minimize the scatter for each class.

What if we have 3 classes?

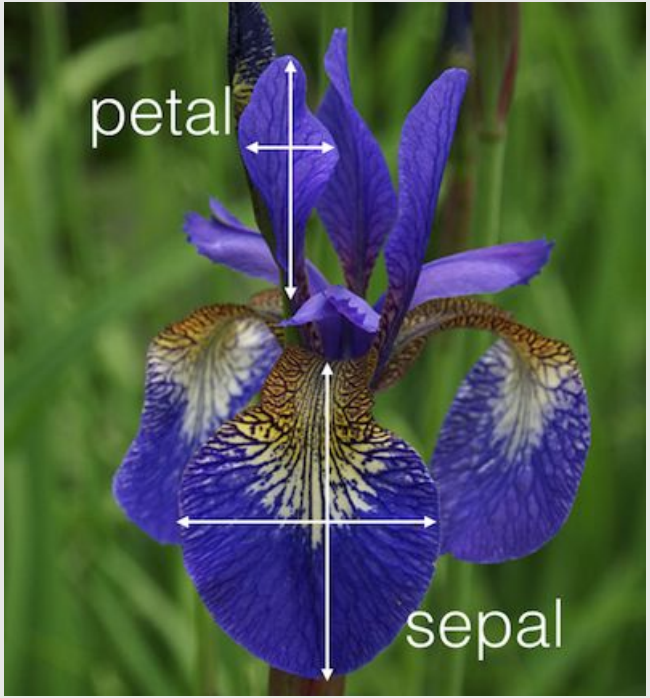


$$\frac{d^2 + d^2 + d^2}{s^2 + s^2 + s^2}$$

LDA in a Nutshell (Eigen Decomposition)

1. Compute the d-dimensional mean vectors for the different classes.
2. Compute the scatter matrices (between-class S_B and within-class S_W).
3. Compute the eigenvectors ($u_1, u_2, \ldots, u_d$) and eigenvalues ($\lambda_1, \lambda_2, \ldots, \lambda_d$) for the scatter matrices $S_W^{-1} S_B$.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors.
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace.

LDA Step by Step



http://sebastianraschka.com/Articles/2014_python_lda.html

150 iris flowers from three different species.

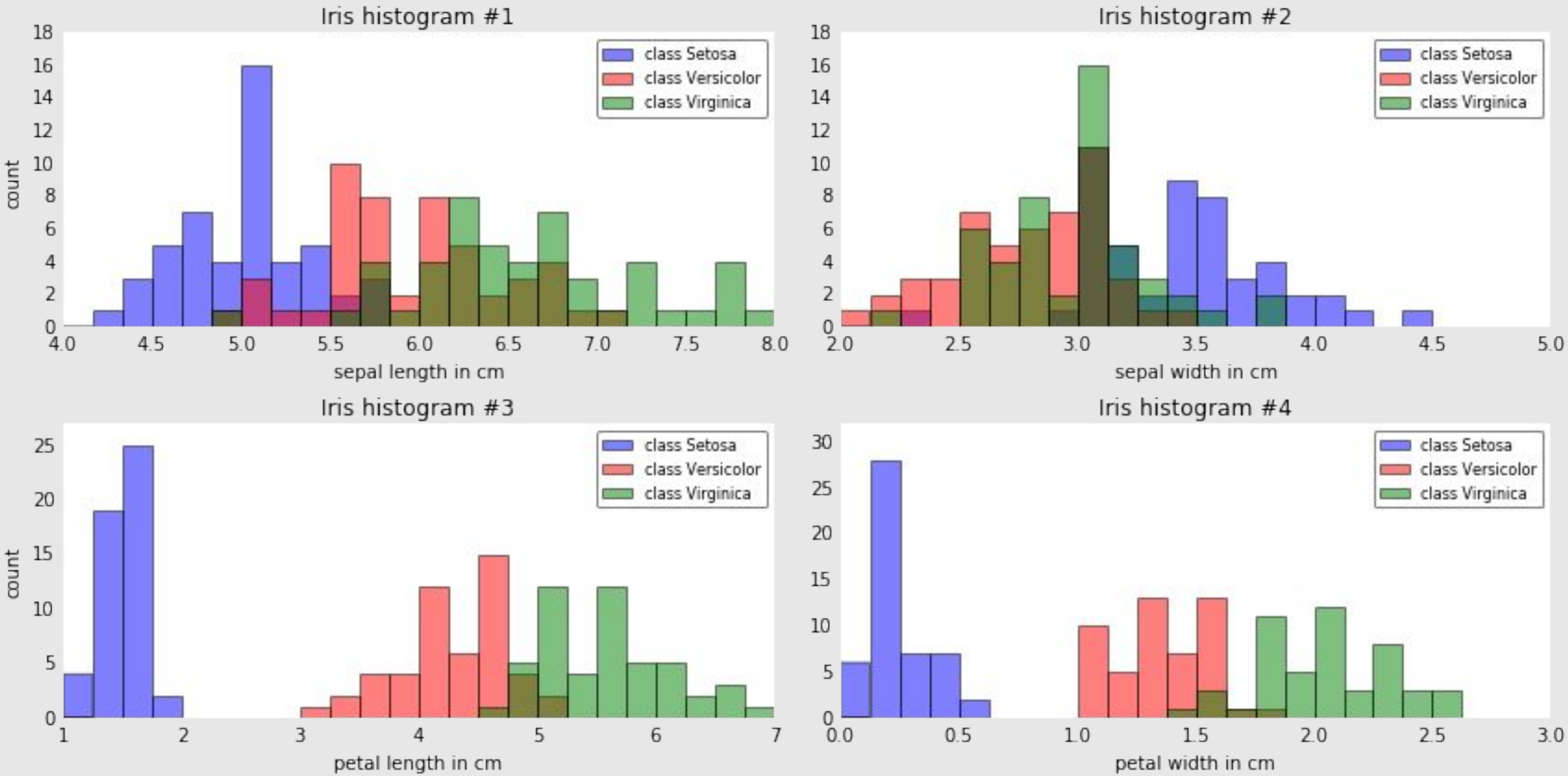
The three classes in the Iris dataset:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

The four features of the Iris dataset:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

LDA Step by Step



LDA Step by Step

1. Compute the d-dimensional mean vectors for the different classes.

LDA Step by Step

1. Compute the d-dimensional mean vectors for the different classes.

$$\mu_1 : [\ 5.01 \quad 3.42 \quad 1.46 \quad 0.24\ ]$$

$$\mu_2 : [\ 5.94 \quad 2.77 \quad 4.26 \quad 1.33\ ]$$

$$\mu_3 : [\ 6.59 \quad 2.97 \quad 5.55 \quad 2.03\ ]$$

LDA Step by Step

2. Compute the **scatter matrices** (between-class S_B and within-class S_W)

   Within-class scatter matrix S_W:

$$S_W = \sum_{i=1}^{c} S_i \text{ , where } S_i = \sum_{x \in D_i}^{n} (x - \mu_i)(x - \mu_i)^T$$

LDA Step by Step

2. Compute the **scatter matrices** (between-class S_B and within-class S_W)

   Within-class scatter matrix S_W:

$$\begin{bmatrix} 38.96 & 13.68 & 24.61 & 5.66 \\ 13.68 & 7.04 & 8.12 & 4.91 \\ 24.61 & 8.12 & 27.22 & 6.25 \\ 5.66 & 4.91 & 6.25 & 6.18 \end{bmatrix}$$

LDA Step by Step

2. Compute the **scatter matrices** (between-class S_B and within-class S_W)

   Between-class scatter matrix S_B:

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^{\mathrm{T}}$$

   where μ is the overall mean, and μ_i and N_i are the sample mean and sizes of the respective classes.

LDA Step by Step

2. Compute the **scatter matrices** (between-class S_B and within-class S_W)

   Between-class scatter matrix S_B:

$$\begin{bmatrix} 63.21 & -19.53 & 165.16 & 71.36 \\ -19.53 & 10.98 & -56.05 & -22.49 \\ 65.16 & -56.05 & 436.64 & 186.91 \\ 71.36 & -22.49 & 186.91 & 80.60 \end{bmatrix}$$

LDA Step by Step

3. Compute the eigenvectors $(u_1, u_2, \ldots, u_d)$ and eigenvalues $(\lambda_1, \lambda_2, \ldots, \lambda_d)$ for the scatter matrices $S_W^{-1} S_B$.

u_1: $\begin{pmatrix} -0.205 \\ -0.387 \\ 0.546 \\ 0.714 \end{pmatrix}$ u_2: $\begin{pmatrix} -0.009 \\ -0.589 \\ 0.254 \\ -0.767 \end{pmatrix}$ u_3: $\begin{pmatrix} 0.179 \\ -0.318 \\ -0.366 \\ 0.601 \end{pmatrix}$ u_4: $\begin{pmatrix} 0.179 \\ -0.318 \\ -0.366 \\ 0.601 \end{pmatrix}$

λ_1: 3.23e+01 λ_2: 2.78e-01 λ_3: -4.02e-17 λ_4: -4.02e-17

LDA Step by Step

4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors.

   Eigenvalues in decreasing order:

   32.27
   0.27
   5.71e-15
   5.71e-15

LDA Step by Step

4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors.

Eigenvalues in decreasing order:

32.27
0.27
5.71e-15
5.71e-15

Variance explained:

λ_1: 99.15%
λ_2: 0.85%
λ_3: 0.00%
λ_4: 0.00%

LDA Step by Step

4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors.

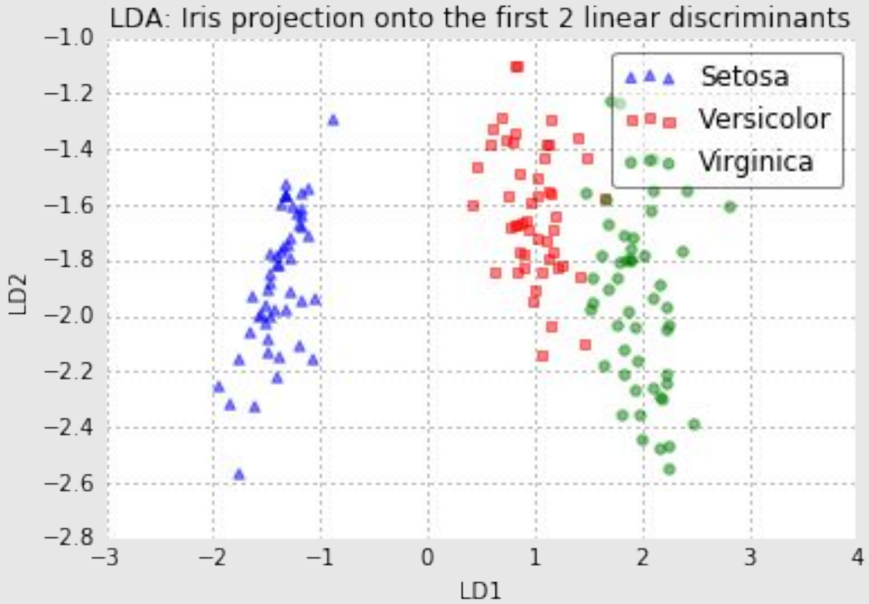
u_1: $\begin{pmatrix} -0.205 \\ -0.387 \\ 0.546 \\ 0.714 \end{pmatrix}$ λ_1: 3.23e+01

u_2: $\begin{pmatrix} -0.009 \\ -0.589 \\ 0.254 \\ -0.767 \end{pmatrix}$ λ_2: 2.78e-01

$$\Rightarrow \begin{pmatrix} -0.205 & -0.009 \\ -0.387 & -0.589 \\ 0.546 & 0.254 \\ 0.714 & -0.767 \end{pmatrix}$$

LDA Step by Step

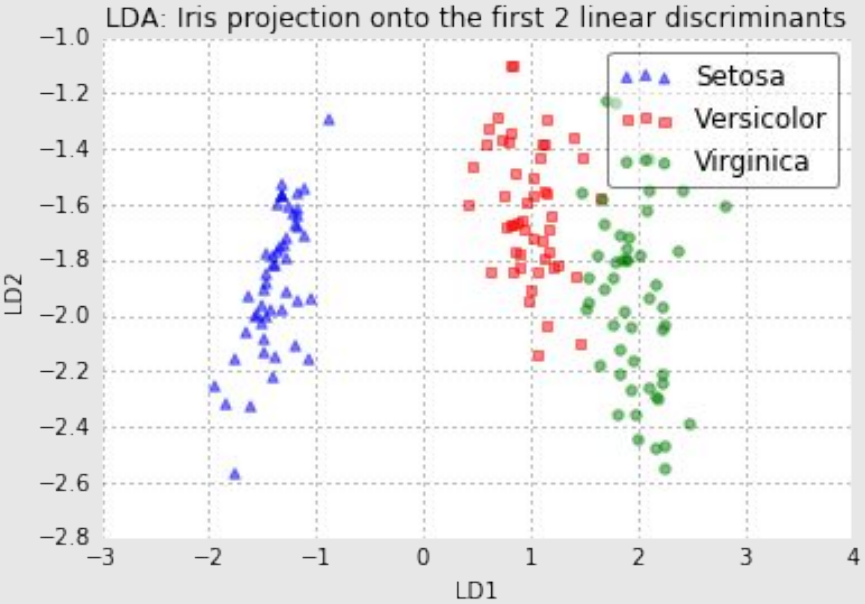
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace.



LDA Step by Step

http://sebastianraschka.com/Articles/2014_python_lda.html

5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace.



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”