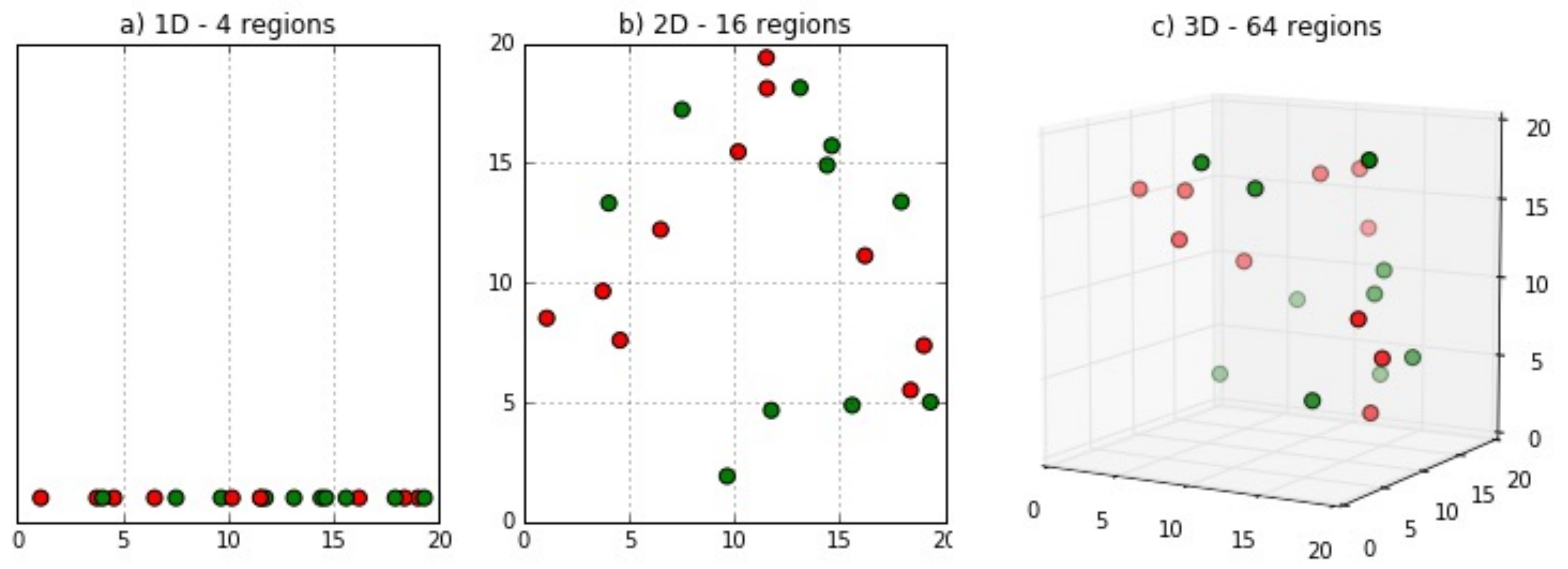
Politecnico di Bari

ML principles

The Curse of Dimensionality

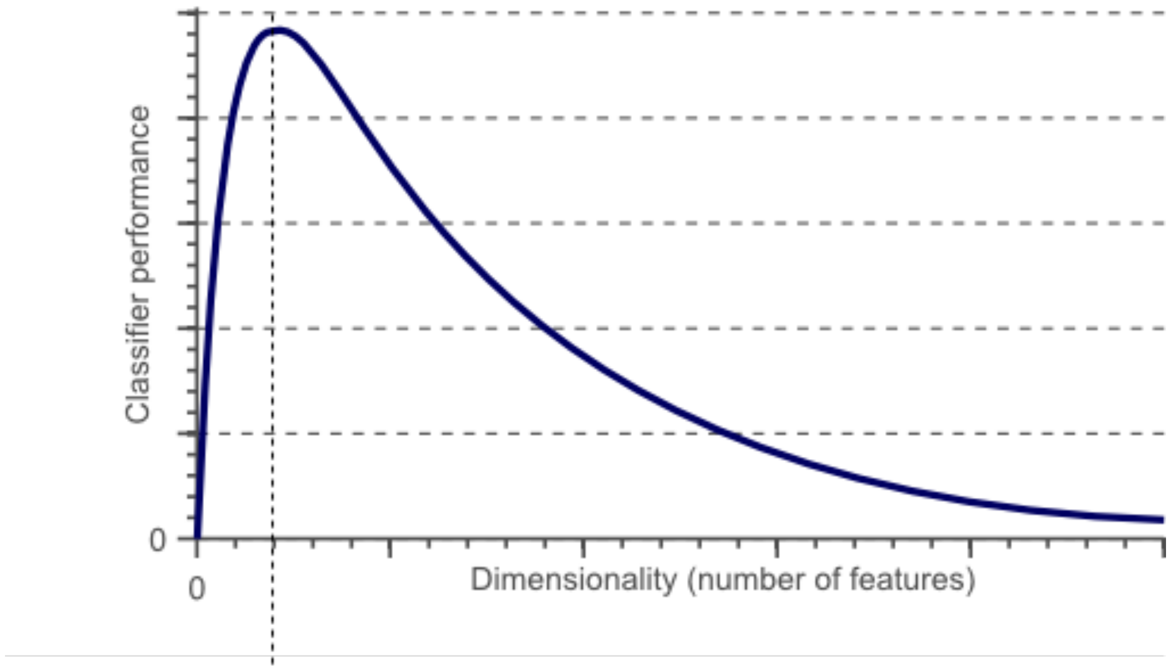
The Curse of Dimensionality



- A single feature is not sufficient to build a perfect classifier (1d).
- Adding a second feature still does not result in a linearly separable classification problem (2d).
- Imagine that adding a third feature results in a linearly separable classification problem in our example (3d).

- Strategy: Can we build a perfect classifier by defining a very big number of features? The answer is no!

The Curse of Dimensionality



Optimal number of features

As the dimensionality increases, the classifier's performance increases until the optimal number of features is reached. Further increasing the dimensionality without increasing the number of training samples results in a decrease in classifier performance.

Why couldn’t we simply increase the number of features?

In the example we built a perfect classifier BUT note how **the density of the training samples decreases exponentially** when we increase the dimensionality of the problem.

- 1d: we almost covered the complete 1d feature space; sample density ~ 30/4.
- 2d: the 2D feature space has an area of 4×4=16 unit squares; sample density ~ 30/16.
- 3d: a feature space volume of 4x4x4 = 64 unit cubes; sample density ~ 30/30.

The Curse of Dimensionality

- If the amount of available training data is fixed, then **overfitting** occurs if we keep adding dimensions.
- The “curse” of dimensionality introduces the **sparseness** of the training data. The more features we use, the sparser the data becomes, and consequently accurate estimation of the classifier’s parameters (i.e., its decision boundaries) becomes more difficult.
- To maintain the same coverage and avoid overfitting, the amount of training data would need to grow **exponentially** fast, but this condition rarely occurs.

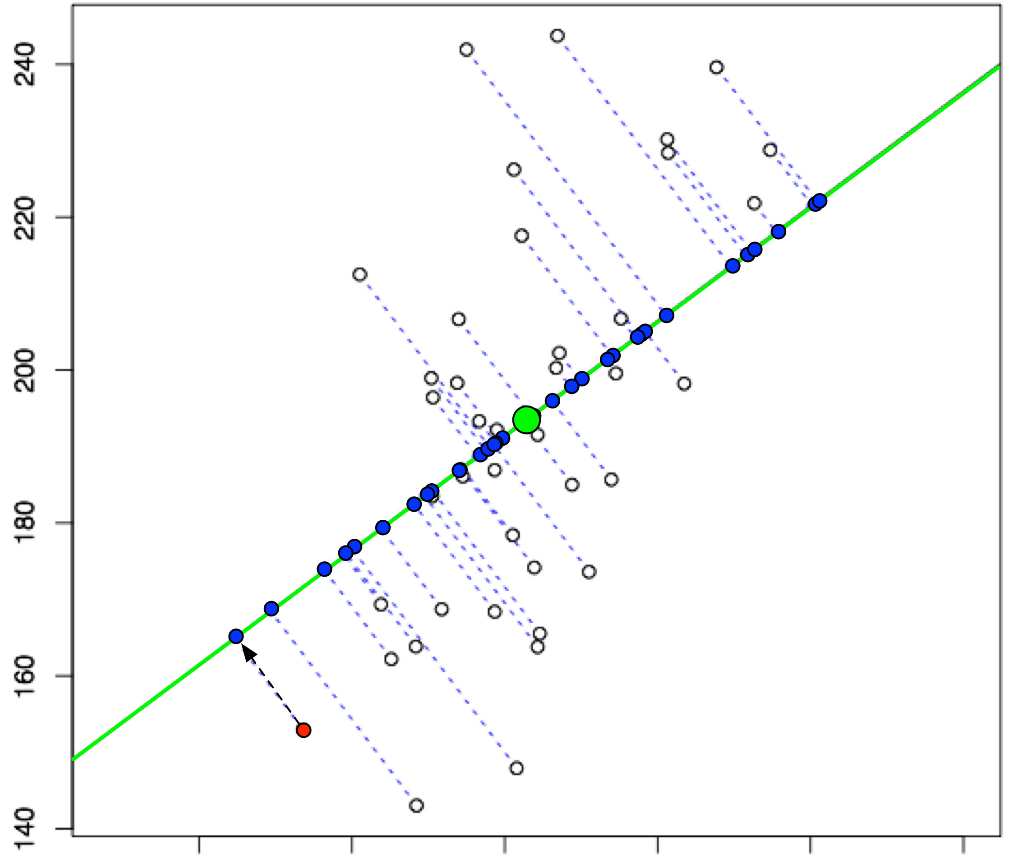
Politecnico di Bari

Unsupervised Learning

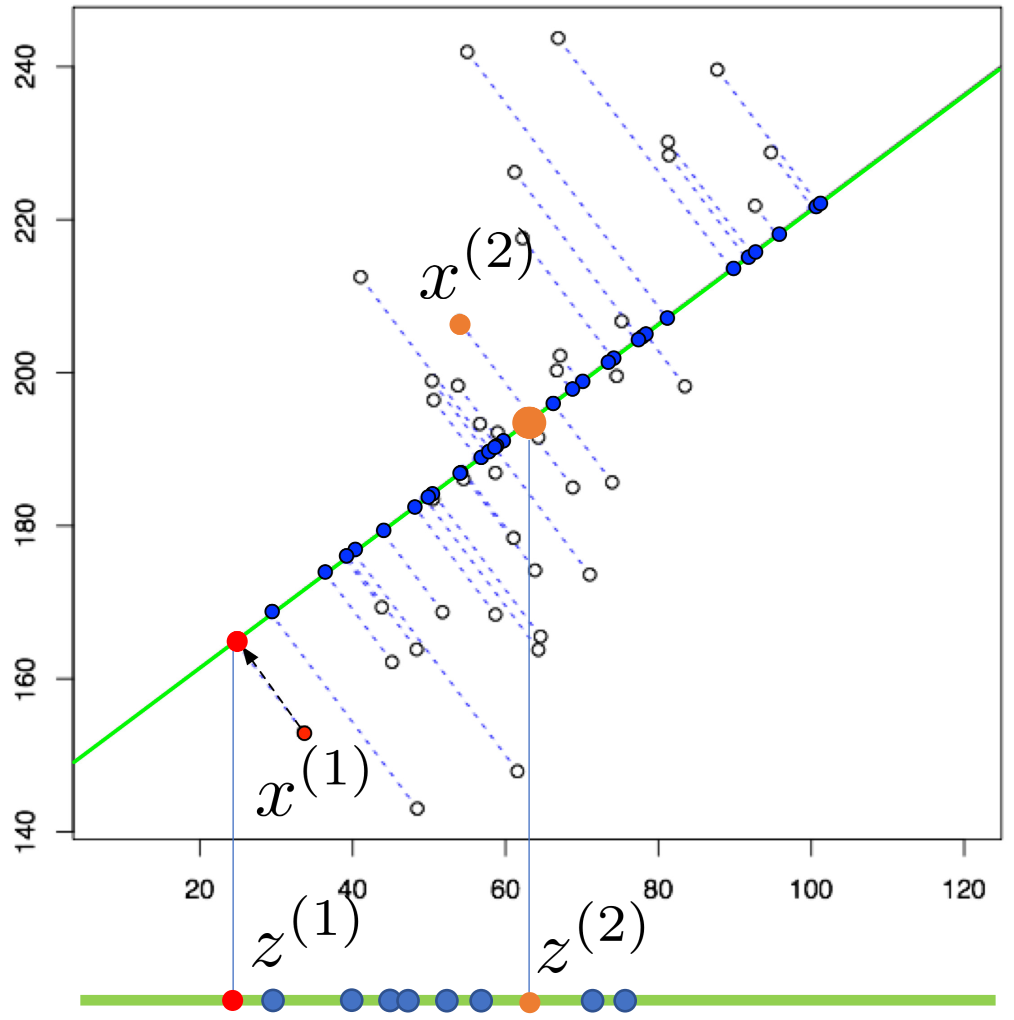
Dimensionality Reduction

Motivation I: Data Compression

Reduce data from 2D to 1D



Motivation I: Data Compression

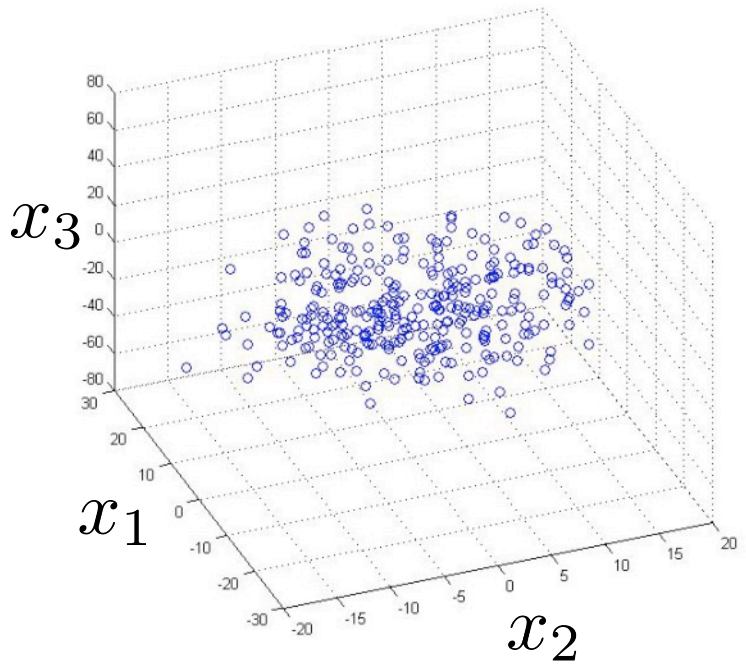


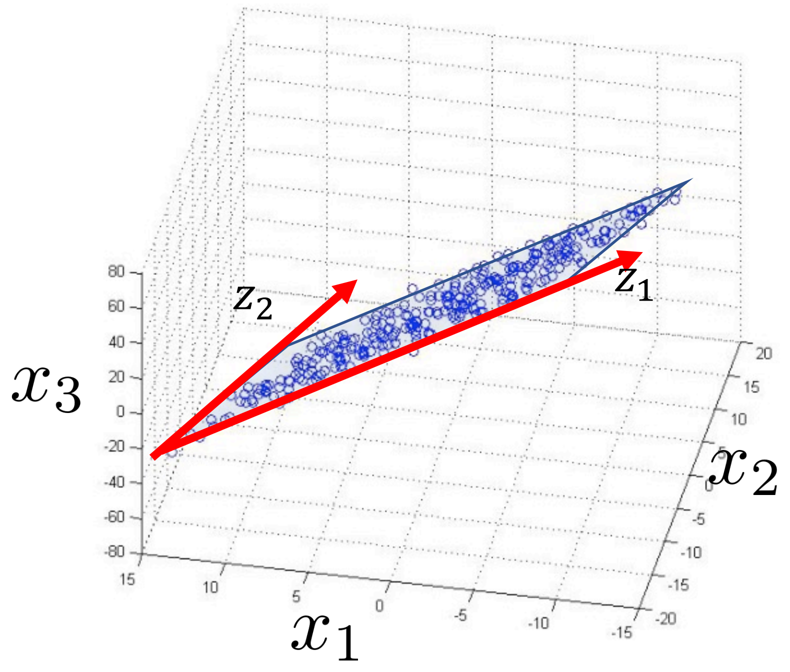
Reduce data from 2D to 1D

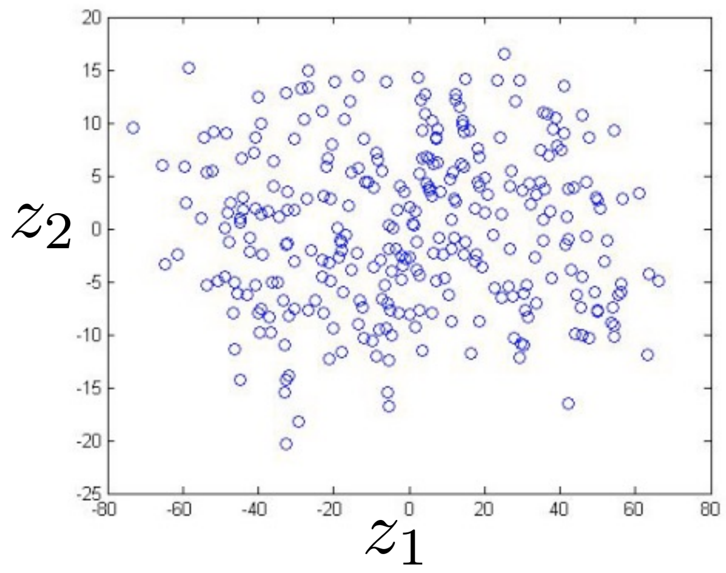
$$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$$
$$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$$
$$\ldots$$
$$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$$

Motivation I: Data Compression

Reduce data from 3D to 2D







$$x^{(i)} \in \mathbb{R}^3 \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \mathbf{z}^{(\mathbf{i})} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

Motivation II: Data Visualization

	x_1	x_2	x_3	x_4	x_5	
Country	Region	Population	Area	Pop. Density	Coastline	…
Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0	0,00	…
Albania	EASTERN EUROPE	3581655	28748	124,6	1,26	…
Algeria	NORTHERN AFRICA	32930091	2381740	13,8	0,04	…
American Samoa	OCEANIA	57794	199	290,4	58,29	…
Andorra	WESTERN EUROPE	71201	468	152,1	0,00	…
Angola	SUB-SAHARAN AFRICA	12127071	1246700	9,7	0,13	…
Anguilla	LATIN AMER. & CARIB	13477	102	132,1	59,80	…
Antigua & Barbuda	LATIN AMER. & CARIB	69108	443	156,0	34,54	…
Argentina	LATIN AMER. & CARIB	39921833	2766890	14,4	0,18	…
…	…	…	…	…	…	…

$$x \in \mathbb{R}^{50}$$

$$x^{(i)} \in \mathbb{R}^{50}$$

Motivation II: Data Visualization

$$z_1 \qquad z_2$$

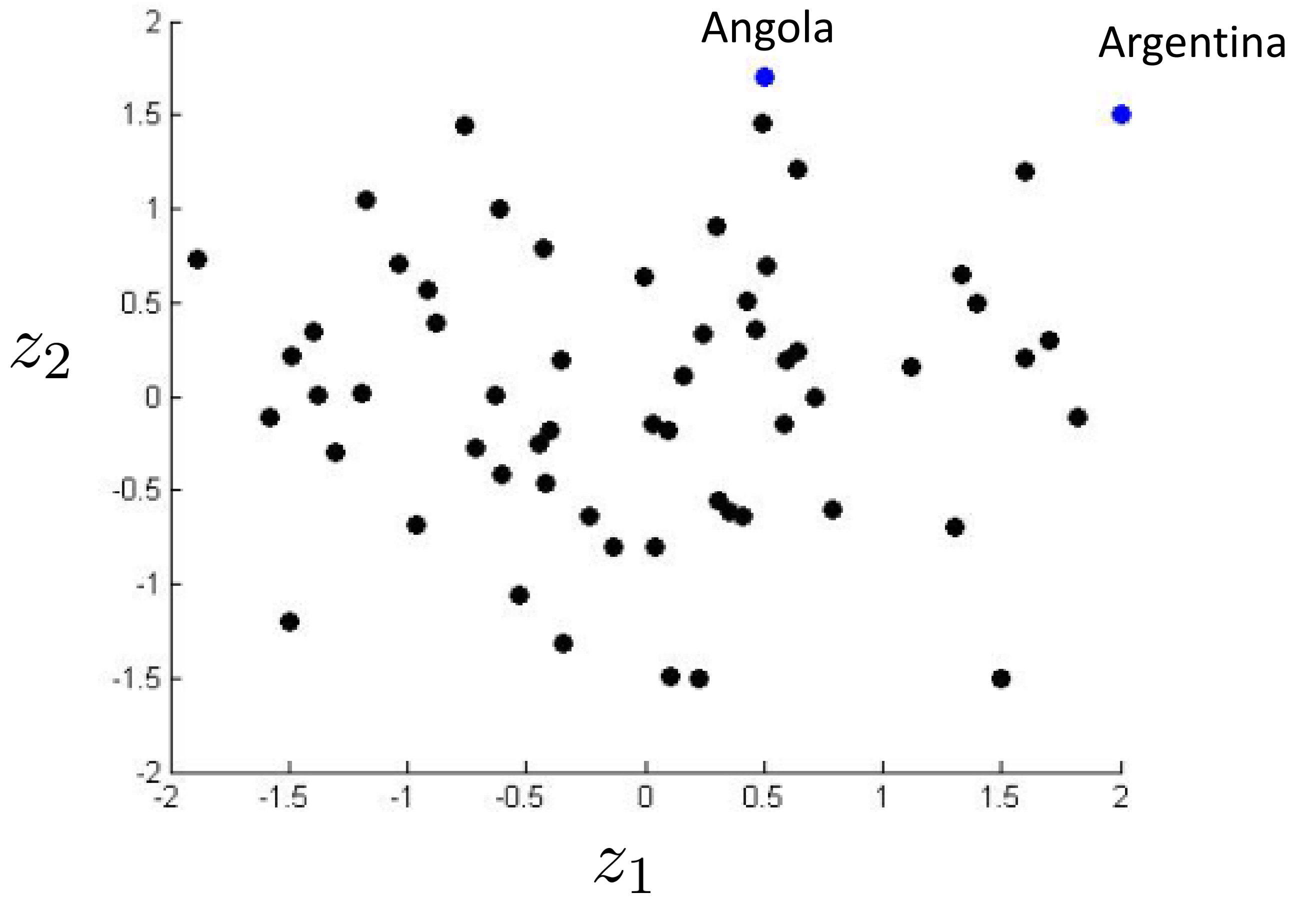
Country	z1	z2
Afghanistan	1.6	1.2
Albania	1.7	0.3
Algeria	1.6	0.2
American Samoa	1.4	0.5
Andorra	0.5	1.7
Angola	2	1.5
Anguilla	1.4	1.5
Antigua & Barbuda	0.5	0.5
Argentina	2	1.7
…	…	…

$$z \in \mathbb{R}^2$$

$$z^{(i)} \in \mathbb{R}^2$$

We reduced data from 50D to 2D

Motivation II: Data Visualization



$$z \in \mathbb{R}^2$$

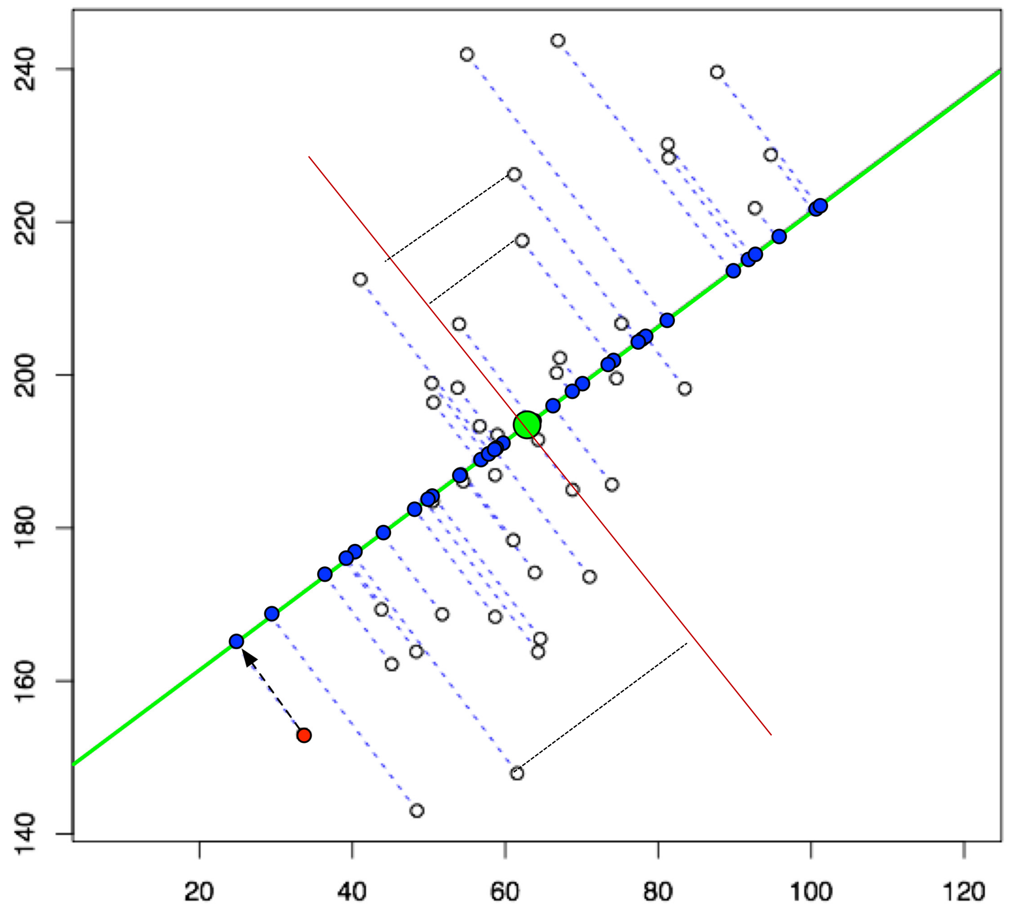
$$z^{(i)} \in \mathbb{R}^2$$

Dimensionality Reduction

Principal Component Analysis

we try to keep as much as possible the variance of the original data. the variance is information, entropy

Problem formulation

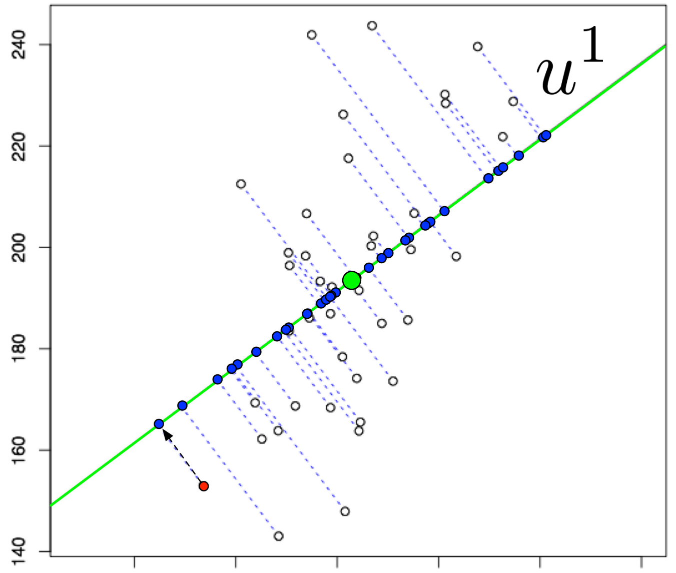


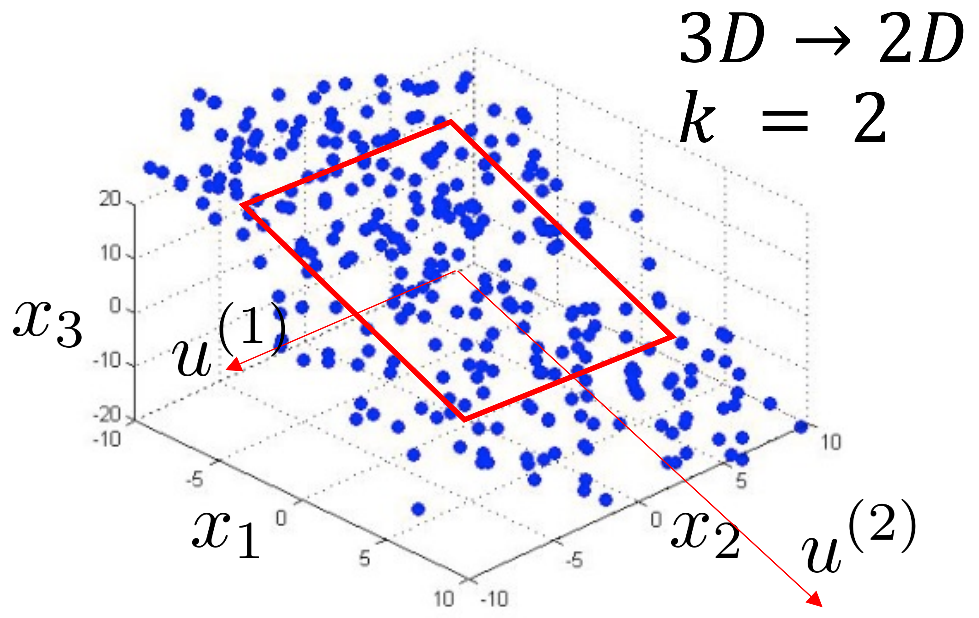
$$z \in \mathbb{R}^2$$

$$z^{(i)} \in \mathbb{R}^2$$

Problem formulation

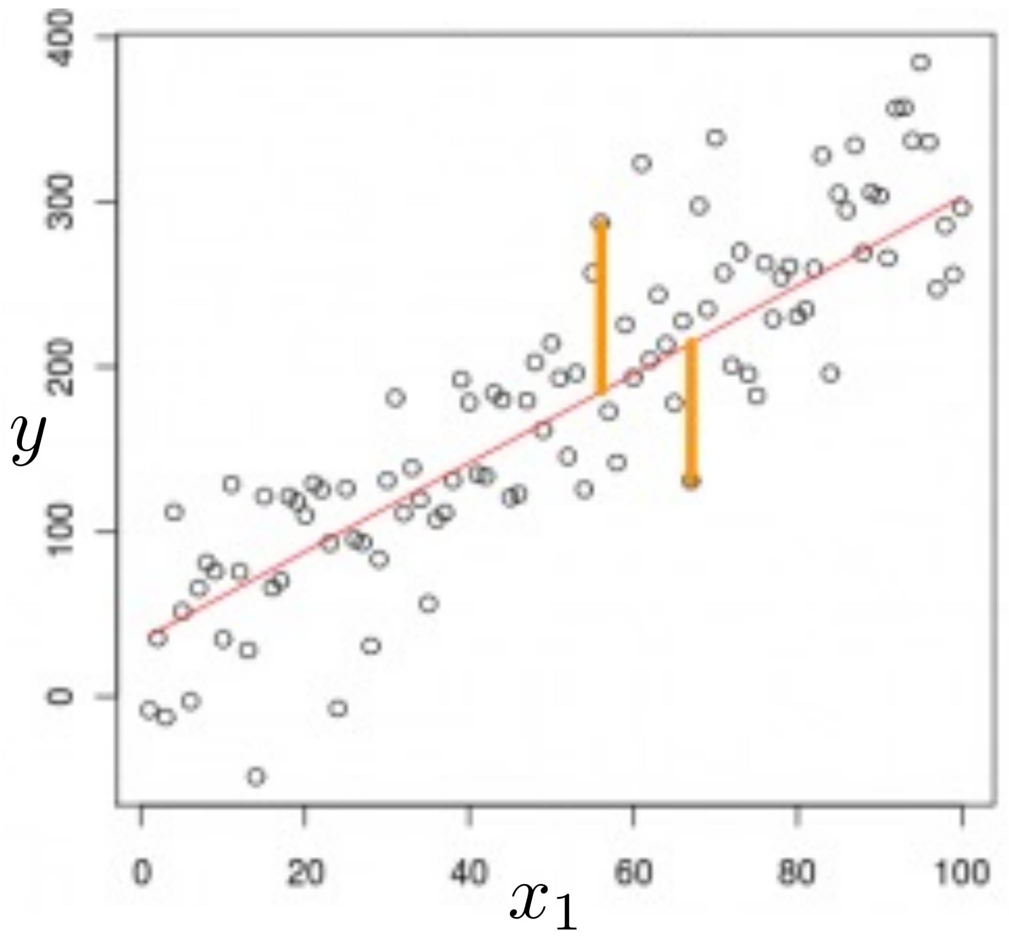
n dimension of original space
k dimension of compressed space

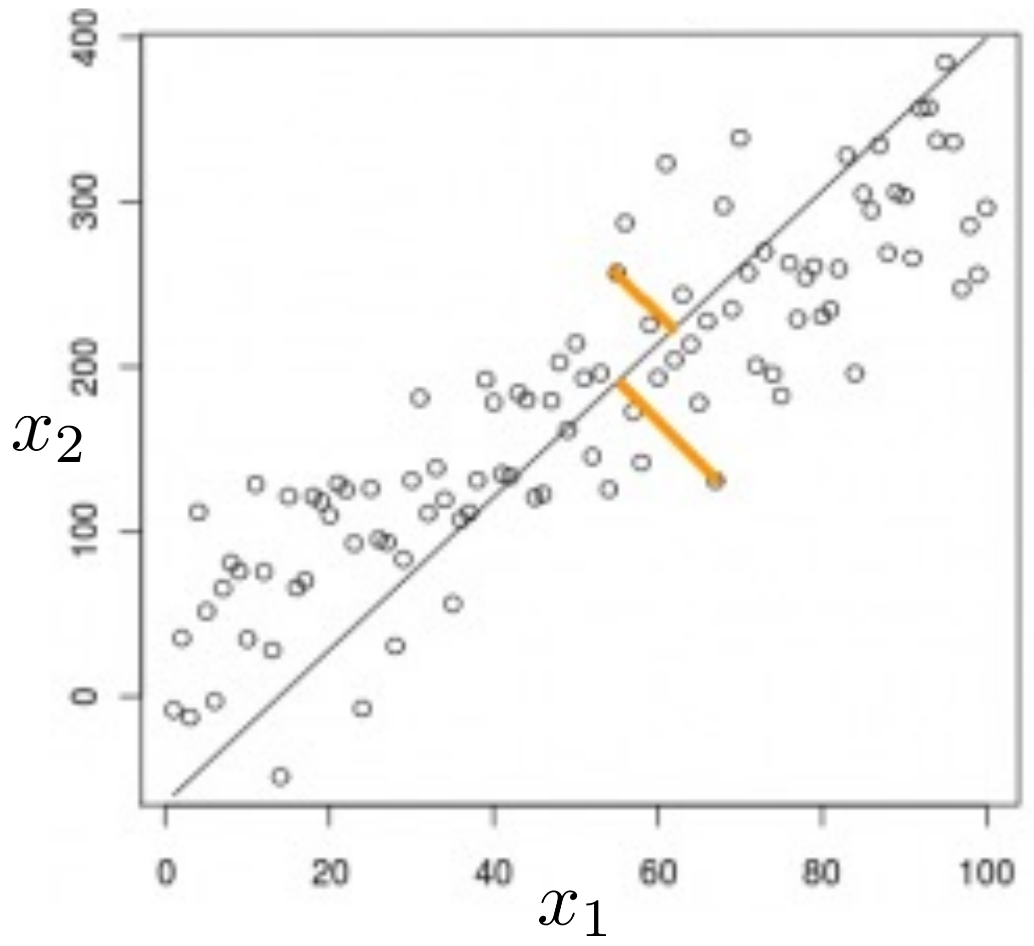




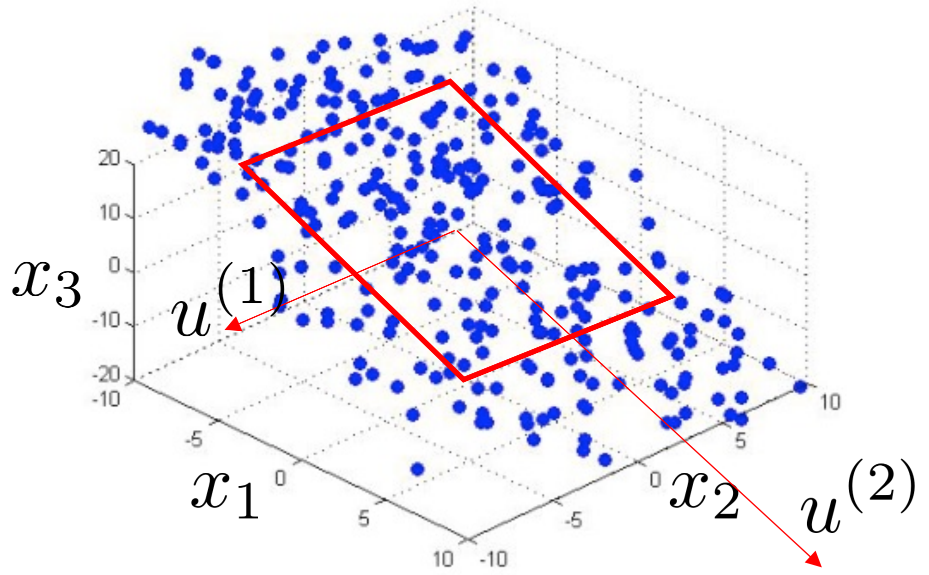
- Reduce from 2D to 1D: Find a direction (a vector u in $\mathbb{R}^n$) onto which to project to minimize the projection error.
- Reduce from nD to kD: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project in order to minimize the projection error.

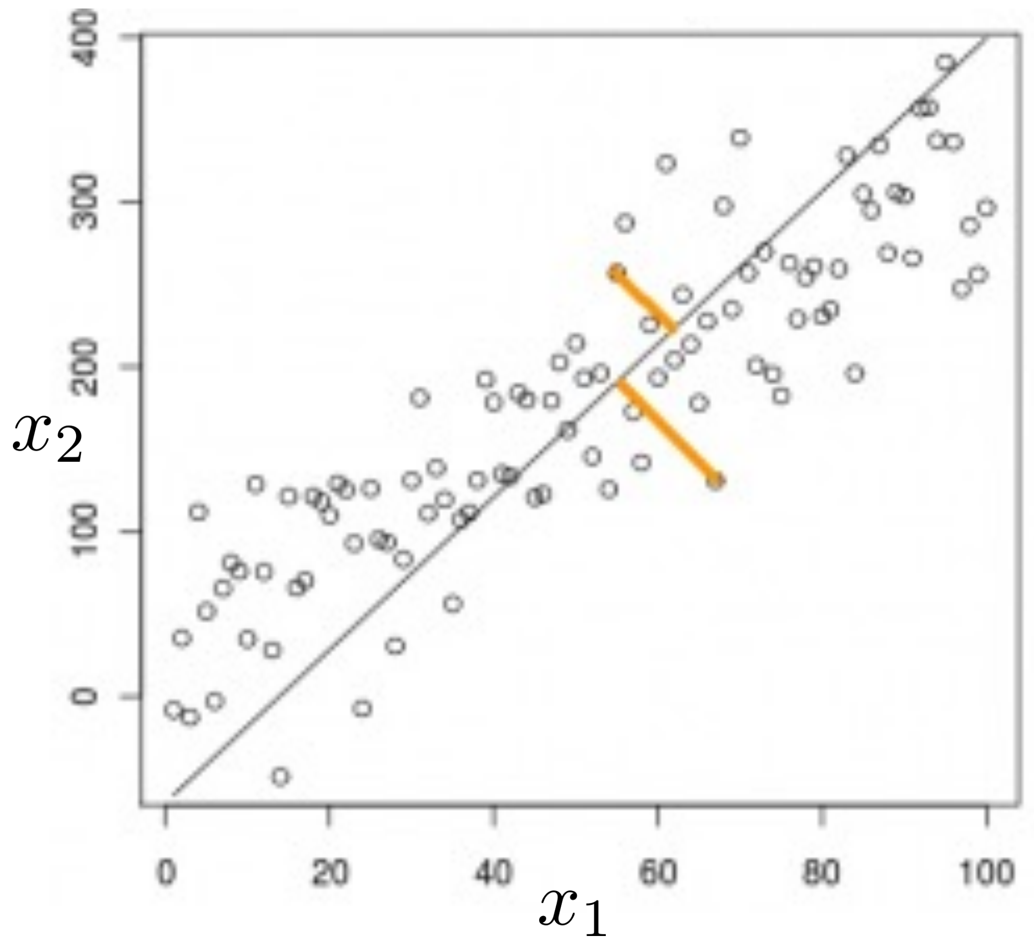
PCA is NOT linear regression





PCA – PROS/CONS





PCA Algorithm summary

1. Preprocess Data
2. Compute covariance matrices
3. Compute svd
4. Transform data in the new space

Data preprocessing

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

Feature scaling/Mean normalization

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

$$x_j^{(i)} \leftarrow x_j^{(i)} - \mu_j$$ (zero mean)

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{stdev_j}$$ (same scale among features)

Compute covariance matrix

Now we can compute the covariance **matrix** between each pair of features. Each component is defined as:

$$\begin{aligned}\Sigma_{ij} &= Cov(x_i, x_j) \\ &= \mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)] = \frac{1}{m} \cdot \sum_{l=1}^{m} \left(x_i^{(l)} - \mu_i\right) \cdot (x_j^{(l)} - \mu_j) \\ &= \mathbb{E}[(x_i)(x_j)]\end{aligned}$$

This operation can be easily vectorized:

$$\Sigma = \frac{1}{m} \sum_{l=1}^{m} \underbrace{\underbrace{x^{(l)}}_{n \times 1} \cdot \underbrace{x^{(l)^T}}_{1 \times n}}_{n \times n}$$

these are equivalent, why?

Singular Value Decomposition

In order to compute the eigenvectors of Σ we can compute the Singular Value Decomposition (SVD):

$$[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \boldsymbol{svd}(\boldsymbol{\Sigma})$$

SVD is a factorization that takes as input a generic matrix A in $C_{m\times n}$ such that

$$\boldsymbol{A} = \boldsymbol{U} \cdot \boldsymbol{S} \cdot \boldsymbol{V}^*$$

Where:

- $\boldsymbol{U}$ is the left eigenvectors matrix, a left unitary matrix (the product with its conjugate transpose is the identity matrix $U \cdot U^* = I$, for real matrices $U^{-1} = U^T$ is valid) of shape $\boldsymbol{m}\times\boldsymbol{m}$
- $\boldsymbol{S}$ is a diagonal matrix of shape $\boldsymbol{m}\times \boldsymbol{n}$ (the first n components are not null and they are the eigenvalues of A, ordered in decreasing order)
- $\boldsymbol{V}^*$ is the conjugate transpose of a right unitary matrix of shape $\boldsymbol{n}\times\boldsymbol{n}$

But we are computing the SVD of a covariance matrix (a symmetric matrix), hence $U = V$

$$\boldsymbol{\Sigma} = \boldsymbol{U} \cdot \boldsymbol{S} \cdot \boldsymbol{U}^{\boldsymbol{T}}$$

s contain eigen value of the original data, U contain the eigen vector
eigen value=autovettori

Singular Value Decomposition

matrix rappresent space transformation

the eigen vector are the principal component of that space

From $[U, S, V] = svd(\Sigma)$ we get:

$$\mathbf{U} = \begin{bmatrix} | & | & & | \\ \underbrace{u^{(1)} & u^{(2)} & \ldots}_{k} & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

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

$$\mathbf{z}^{(\mathbf{i})} = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \ldots & u^{(k)} \\ | & | & & | \end{bmatrix}}_{\substack{n \times k \\ Ureduce}}^{T} \in \mathbb{R}^{n \times k}$$

$$z \in \mathbb{R}^k$$

$$\mathbf{x}^{(\mathbf{i})} = \underbrace{\underbrace{\begin{bmatrix} \text{——} & (u^{(1)})^T & \text{——} \\ & \vdots & \\ \text{——} & (u^{(k)})^T & \text{——} \end{bmatrix}}_{k \times n} \underbrace{x^{(i)}}_{n \times 1}}_{k \times 1}$$

PCA Algorithm summary

1. Preprocess Data
2. Compute Sigma

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

3. Compute svd

$$[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \boldsymbol{svd}(\boldsymbol{\Sigma})$$

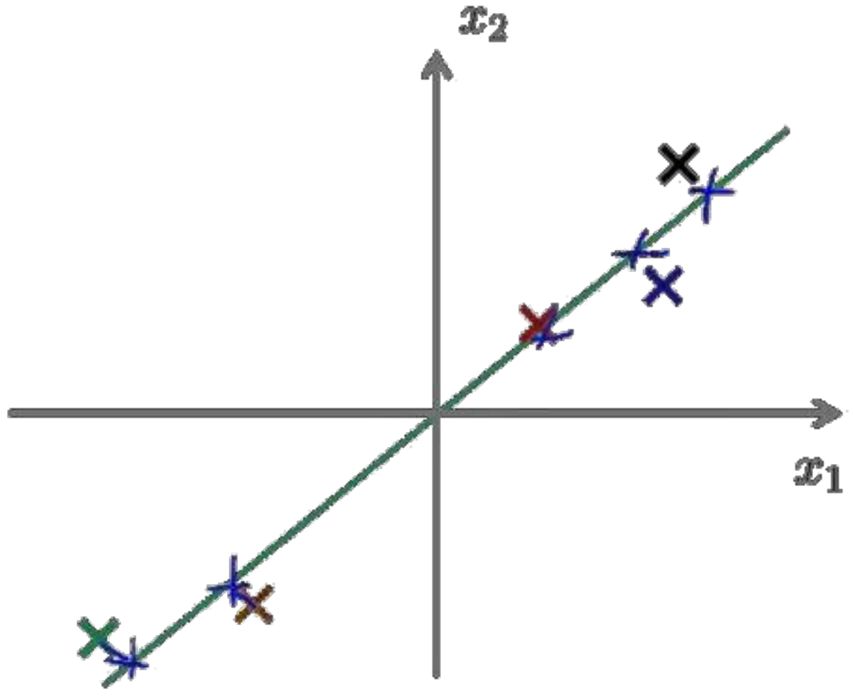
4. Choose the number of dimension and extract the eigenvectors

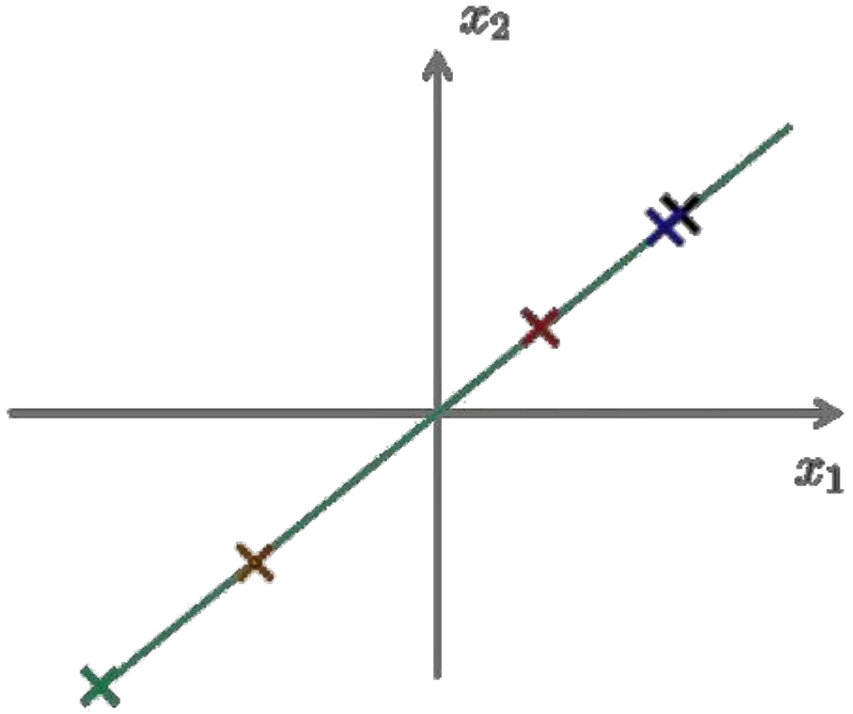
   Ureduce = U(:,1:k)

5. Map data in the new space

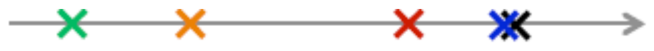
   z = Ureduce^T*x

Reconstruction from compressed representation





$$z = Ureduce^T x$$



$$z \in \mathbb{R} \rightarrow x \in \mathbb{R}^2$$

$$\underbrace{Xapprox^{(i)}}_{n \times 1} = \underbrace{\underbrace{Ureduce}_{n \times k} \cdot \underbrace{z^{(i)}}_{k \times 1}}_{n \times 1}$$

Choosing the number of Principal Components

Average squared projection error: $\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

It is usual to choose k to be the smallest value so that

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

99% of variance is retained

Choosing the number of Principal Components

Algorithm:

Try PCA with k=1,2,3…

Compute:

Ureduce,z(1),z(2),…,z(m)

Xapprox(1),…,xapprox(m)

Check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - x^{(i)}_{approx}||^2}{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2} \le 0.01 \longrightarrow$$

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

$$\mathbf{S} = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & \ddots & \\ & & & S_{nn} \end{bmatrix}$$

for a given k

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \le 0.01 \quad \Rightarrow \quad \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \ge 0.99$$

PCA – PROS/CONS

Pros

- Can handle varying densities
- The algorithm is stable over runs and parameter choices
- Robust to outliers

Cons

- Some important parameters must be set by hand

until now we have done linear operation. if i want to cluster something reducing dimensionality, if the data is not linearly separatable, pca is not great and we can use the kernel trick