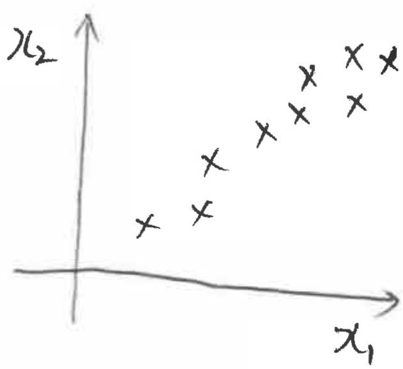
Machine learning Class Note.

Reference : ① Prof. Andrew Ng's Machine Learning Class @ Coursera Standford
② Wikipedia

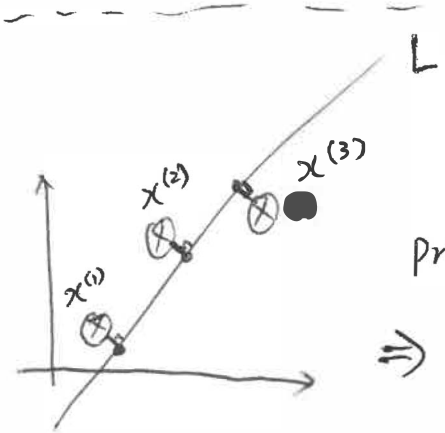
Topic 5 : Principle Component Analysis (PCA)

Data compression :



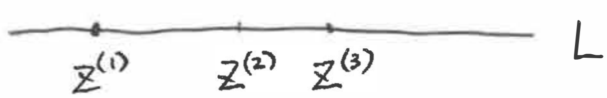
Reduce data from 2D to 1D

$$x^{(1)} \in \mathbb{R}^2 \longrightarrow z^{(1)} \in \mathbb{R}^1$$
$$x^{(2)} \in \mathbb{R}^2 \longrightarrow z^{(2)} \in \mathbb{R}^1$$
$$\vdots$$
$$x^{(m)} \in \mathbb{R}^2 \longrightarrow z^{(m)} \in \mathbb{R}^1$$

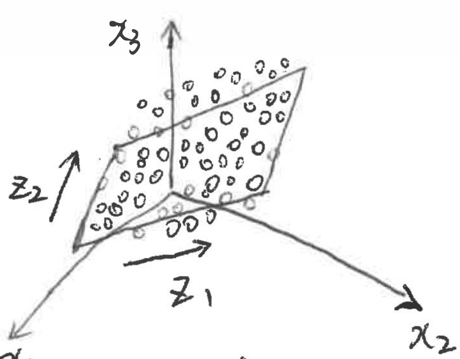


Project the 2D data to the linear line (1D)

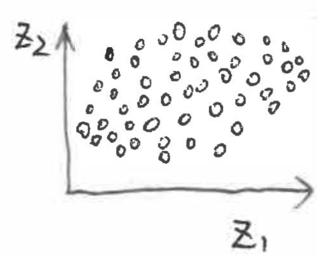
$\Rightarrow$



Data Compression : Reduce data from 3D to 2D



$\Rightarrow$



$x^{(i)} \in \mathbb{R}^3$

$$z^{(i)} \in \mathbb{R}^2 \qquad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

Data compression: Reduce data from 10,000D to 100D

In the same way: project the high-dimentional data to a low-dimentional space.

For example: (100 features for each country)

Country	GDP	area	life expectancy	Proverty index	average income	...
Canada	⋮	⋮	⋮	⋮	⋮	...
China						
India						
Russia						
⋮						

⇓ data compression (from 100D to 3D)

country	Z_1	Z_2	Z_3
Canada	⋮	⋮	⋮
China			
India			
Russia			
⋮			

It is useful for:

① Reduce the data size and computation cost

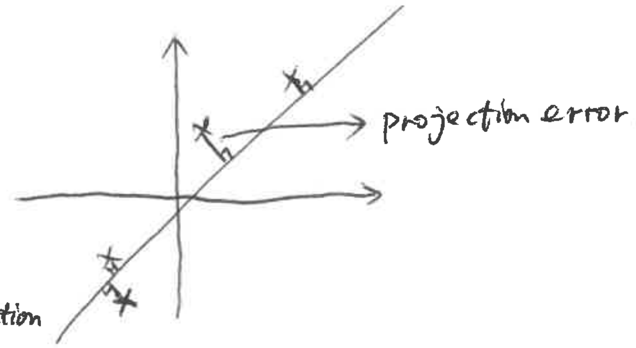
② Data Visualization

③ also keep the main structure of the original data

PCA: commonly used in machine learning and data science.

Example: reduce 2D data to 1D data

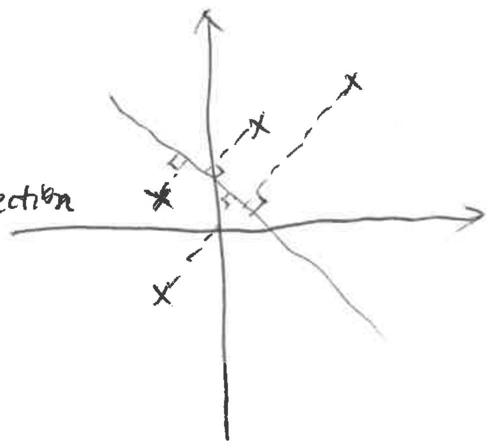
(1)



Good projection

PCA: Find a low-dimentional surface/~~space~~ subspace to project the original high-dimentional data so as to minimize the sum of squares of the projection errors.

(2)

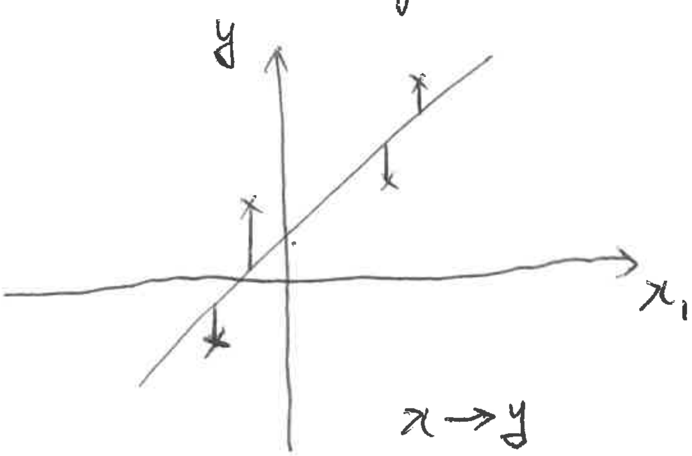


Bad projection

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

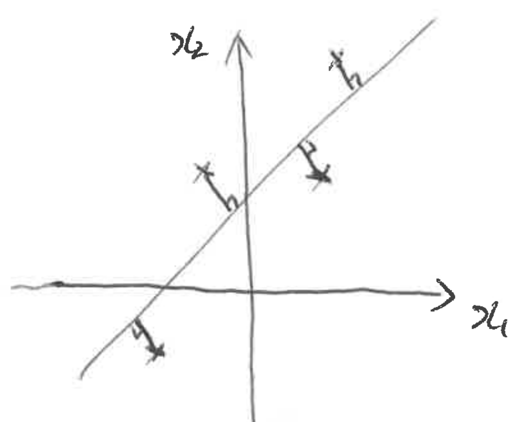
* Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

PCA is not linear regression



$x \to y$

supervised learning

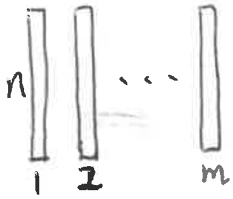


$x_1, x_2, \dots, x_n \to z_1, z_2, \dots, z_k$

unsupervised dimention reduction

data format: m samples, each data sample has n-dimension features.

$x^{(i)} : n \times 1$



$\Rightarrow n \times m$ matrix.

PCA algorithm to reduce data from n-dimension to k-dimension

① compute the "covariance matrix" (Σ : sigma)

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\left(x^{(i)}\right)}_{n\times 1} \cdot \underbrace{\left(x^{(i)}\right)^T}_{1\times n} \Rightarrow n \times n \text{ matrix}$$

② compute "eigenvectors" of matrix Σ:

$n \times n$ matrix

$[u, s, v] = \text{svd}(\text{sigma})$ or $\text{eig}(\text{sigma})$

svd → "singular value decomposition"

$$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & & | \end{bmatrix} \Rightarrow n \times n \text{ matrix},$$

sorted by eigen values in descending order.

$U \in \mathbb{R}^{n \times n}$

☆ Idea here

Eigen values/vectors having larger magnitude are the principle ~~compon~~ components.

(Dropping 'less informative' eigen values/vectors)

③ Choose first k columns from U as U_{reduce}:

$$U_{reduce} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix} \Rightarrow n \times k \text{ matrix}$$

$U_{reduce} \in \mathbb{R}^{n \times k}$

④ Project the original data to the new subspace: $\underbrace{z^{(i)}}_{k\times 1} = \underbrace{U_{reduce}^T}_{k\times n} \cdot \underbrace{x^{(i)}}_{n\times 1}$

PCA algorithm summary (matlab version):

After mean normalization (ensure every feature has zero means) and optionally feature scaling:

$$\underbrace{X}_{m\times n \text{ matrix}} = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

$$Sigma = \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)}\right)\cdot\left(x^{(i)}\right)^T \longrightarrow \underbrace{Sigma}_{n\times n \text{ matrix}} = (1/m) * X' \cdot X;$$

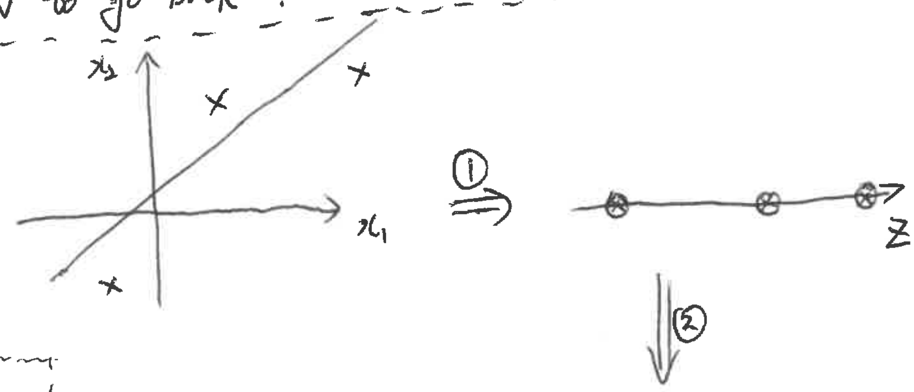
```
[U, S, V] = svd(sigma);
Ureduce = U(:, 1:k);
z = Ureduce' * x;    % ' means transpose in matlab.
```

Reconstruction from compressed representation

$$\text{High dimention data} \xrightarrow[\text{PCA}]{\text{reduce}} \text{low dimention data}$$

how to go back?

① $\underbrace{z}_{k\times 1} = \underbrace{U_{reduce}^T}_{k\times n} \cdot \underbrace{x}_{n\times 1}$

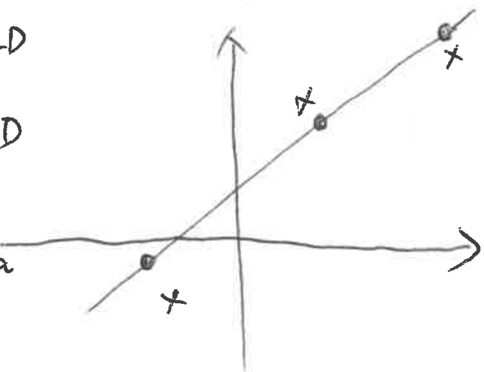


② $\underbrace{x_{approx}^{(i)}}_{n\times 1} = \underbrace{U_{reduce}}_{n\times k} \cdot \underbrace{z^{(i)}}_{k\times 1}$

x: original data in 2D

⊗: projected data in 1D

o: reconstructed data in 2D



Choosing K for PCA: [for after-class reading and search by yourself]

$[u, s, v] = svd(sigma)$

Pick smallest value of K for which:

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1} S_{ii}} \geq 0.99 \quad \text{or } 0.95$$

(99% of variance retained)

How to use PCA?

① Supervised learning speedup.

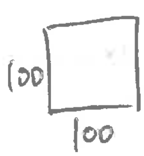
$x^{(1)}, y^{(1)}$
$x^{(2)}, y^{(2)}$
$\vdots$
$x^{(m)}, y^{(m)}$

Exact inputs:

Unlabed dataset: $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$

$\downarrow$ PCA

$z^{(1)}, z^{(2)}, \ldots, z^{(m)} \in \mathbb{R}^{100}$

$x^{(i)} =$  100 × 100 Gray image $\xrightarrow{PCA}$ 100D data.

10,000 features for one image

New training set: $z^{(1)}, y^{(1)}$
$z^{(2)}, y^{(2)}$
$\vdots$
$z^{(m)}, y^{(m)}$

* learn $h_\theta(z)$ $\Downarrow$ Then, learn a machine learning model mapping z to y.

Note: Mapping $x^{(i)}$ to $z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets.

PCA applications: ① data compression $\begin{cases} \text{reduce memory/disk needed to store data.} \\ \text{speed up learning algorithm} \end{cases}$

② visualization ($k=2$ or 3).

Suggestion to use PCA:

Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that does not do what you want, then implement PCA and consider using $z^{(i)}$.