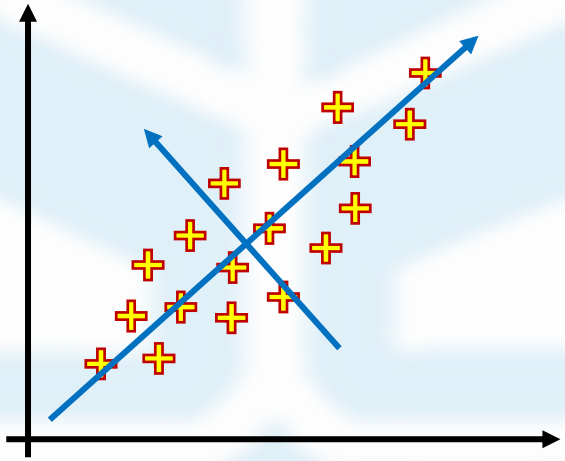
به نام او، به یاد او، برای او

Machine Learning

Dimensionality Reduction

using

Principal Component Analysis(PCA)



Dr. *Ali Valinejad*

valinejad.ir

valinejad@umz.ac.ir

University of Mazandaran

Outline

- **Motivation**
  - **Data Compression**
  - **Data Visualization**
- **Methods:**
  - **Principal Component Analysis(PCA) Problem Formulation**
    - **Principal Component Analysis Algorithm**
    - **Choosing the Number of Principal Components**
    - **Kernel PCA**

University of Mazandaran

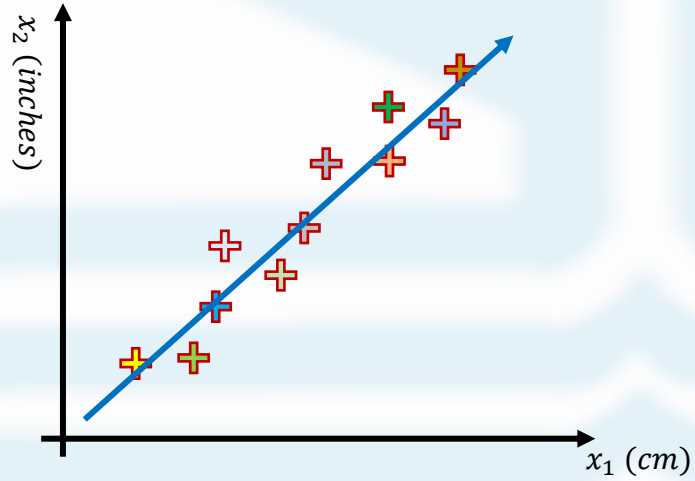
Motivation

Data Compression:

- If we have a lot of *redundant data*, we may want to reduce the dimension of our *features*.
- Doing dimensionality reduction will reduce the total data we have to store in computer memory and will speed up our learning algorithm.

> ***Note***: *In dimensionality reduction, we are reducing our features rather than our number of examples. Our variable will stay the same size, but the length of each vector for each feature will be reduced.*

Example 1: Reduce data from 2D to 1D



https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

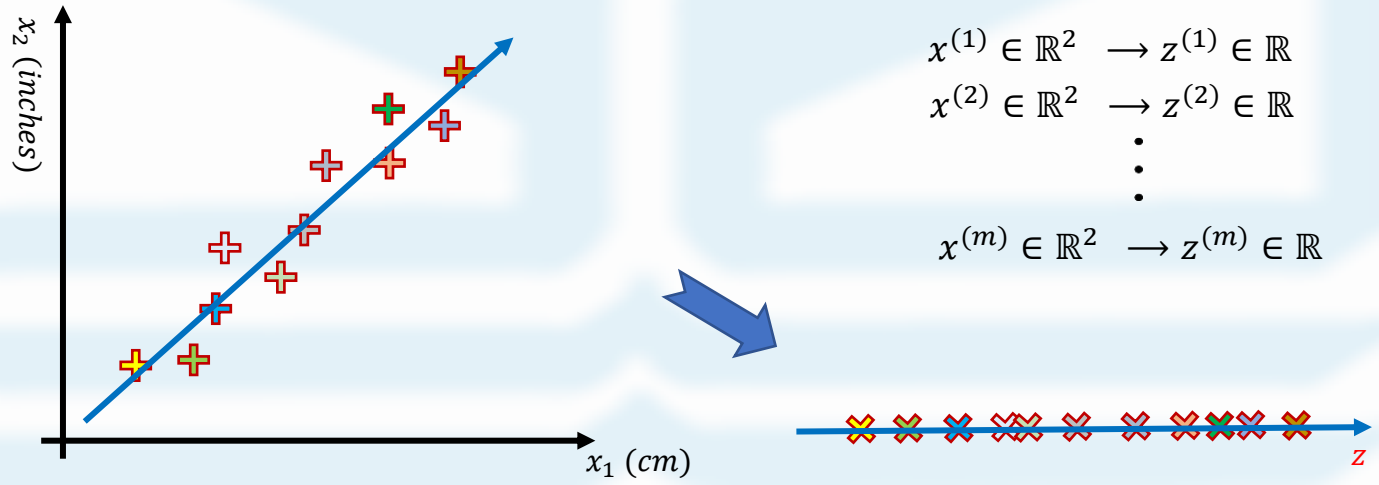
Motivation

Data Compression:

- If we have a lot of *redundant data*, we may want to reduce the dimension of our *features*.
- Doing dimensionality reduction will reduce the total data we have to store in computer memory and will speed up our learning algorithm.

Note: *In dimensionality reduction, we are reducing our features rather than our number of examples. Our variable will stay the same size, but the length of each vector for each feature will be reduced.*

Example 1: Reduce data from 2D to 1D



https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

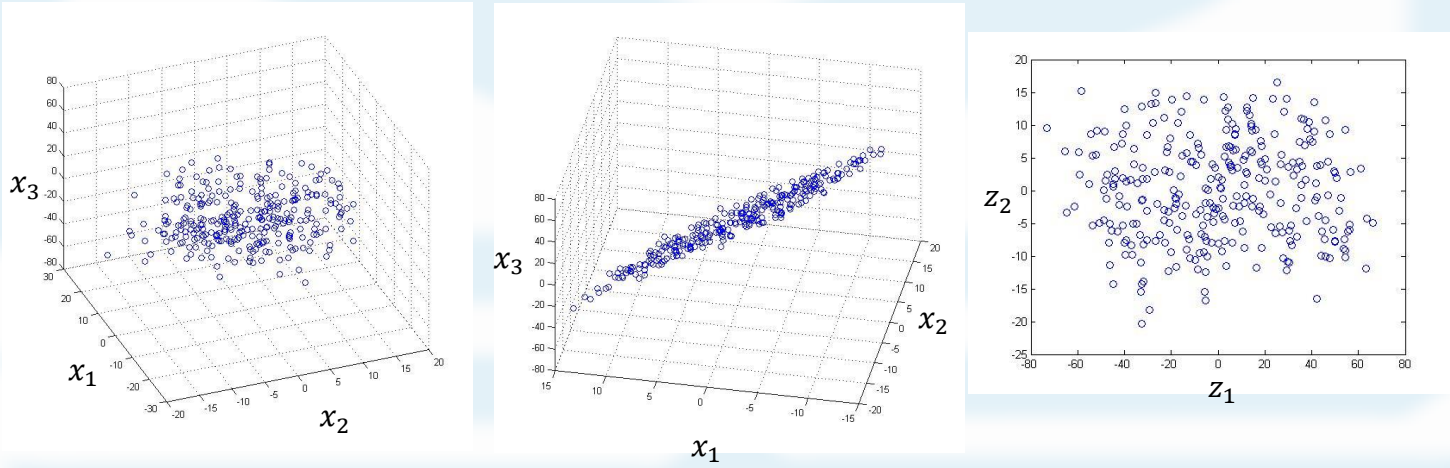
Motivation

Data Compression:

- If we have a lot of *redundant data*, we may want to reduce the dimension of our *features*.
- Doing dimensionality reduction will reduce the total data we have to store in computer memory and will speed up our learning algorithm.

Note: *In dimensionality reduction, we are reducing our features rather than our number of examples. Our variable will stay the same size, but the length of each vector for each feature will be reduced.*

Example 2: Reduce data from 3D to 2D



https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

Motivation

Data Visualization:

- How can we visualize the measurements?

Country	GDP (trillions of US$)	Per capita GDP (thousands of intl. $)	Human Develop-ment Index	Life expectancy	Poverty Index (Gini as percentage)	Mean household income (thousands of US$)	…
Canada	1.577	39.17	0.908	80.7	32.6	67.293	…
China	5.878	7.54	0.687	73	46.9	10.22	…
India	1.632	3.41	0.547	64.7	36.8	0.735	…
Russia	1.48	19.84	0.755	65.5	39.9	0.72	…
Singapore	0.223	56.69	0.866	80	42.5	67.1	…
USA	14.527	46.86	0.91	78.3	40.8	84.3	…
…	…	…	…	…	…	…	

❖ When there are more than three features, it is difficult to see the correlations between the features.

https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

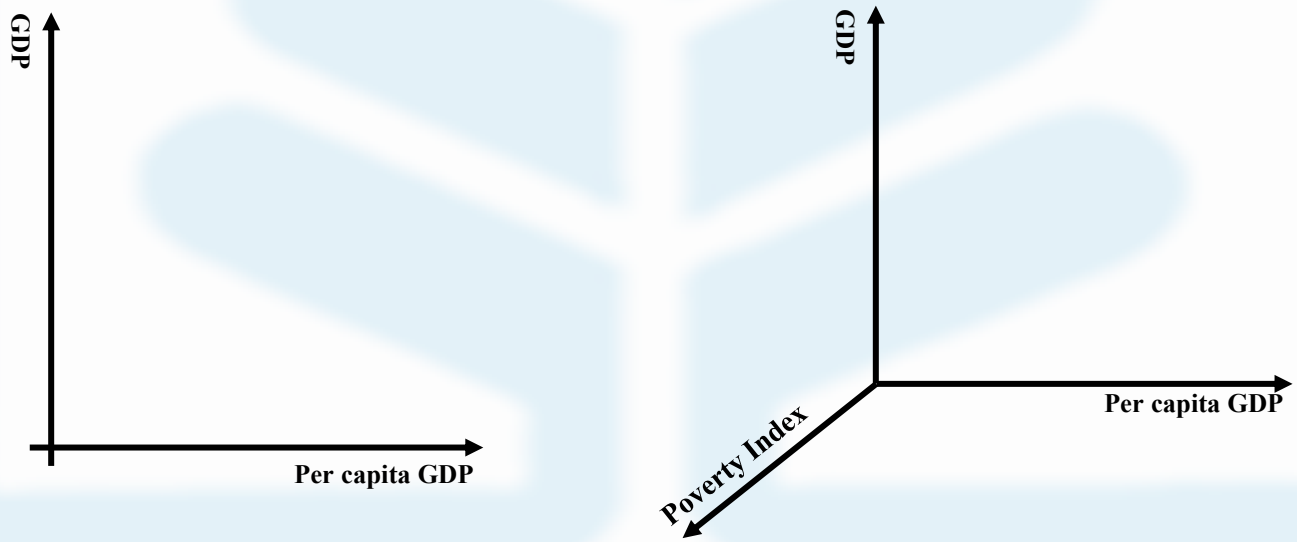
University of Mazandaran

Motivation

Data Visualization:

- Coordinate axes representation is a way to visualize data that is less than or equal three dimensions.
- It is not easy to visualize data that is more than three dimensions.

Is there a representation better than the coordinate axes?



- How can we visualize the other variables?

- Difficult to see in 4 or higher dimensional spaces.

https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

Motivation

Data Visualization:

- It is not easy to visualize data that is more than three dimensions. We can reduce the dimensions of our data to three or less in order to plot it. To do this, We need to *find new features*, (z_1, z_2 and perhaps z_3) that can effectively summarize all the other features.

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
…	…	…

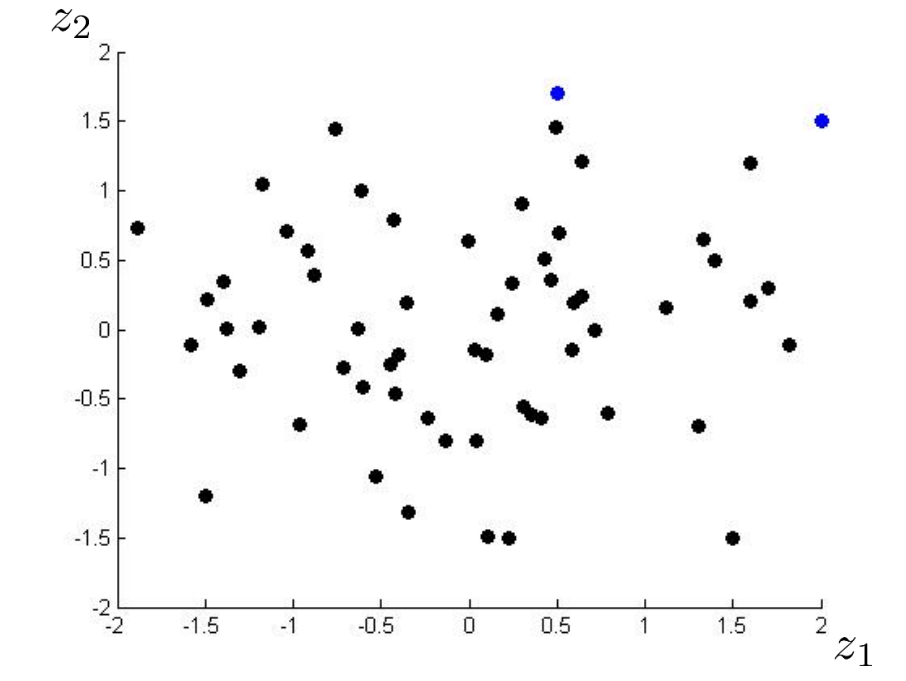
https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

Motivation

Data Visualization:

- It is not easy to visualize data that is more than three dimensions. We can reduce the dimensions of our data to 3 or less in order to plot it. To do this, We need to *find new features*, (z_1, z_2 and perhaps z_3) that can effectively summarize all the other features.



https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

Dimensionality Reduction Methods

Principal Component Analysis(PCA)

University of Mazandaran

❖ Principal Component Analysis(PCA) Problem Formulation:

The PCA method is a statistical method for ***Feature Selection*** and ***Dimensionality Reduction***.

Feature Selection is a process whereby a ***data space*** is transformed into a ***feature space***. In principal both spaces have the same dimensionality. However, in the PCA method, the transformation is design in such way that the data set be represented by a reduced number of "effective" features and yet retain most of the intrinsic information contained in the data; in other words the data set undergoes a dimensionality reduction.

University of Mazandaran

PCA Algorithm

Principal Component Analysis(PCA)

Input: Data points in a n-dimensional space, $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ for i=1,2,…, m

Output:

- project into lower dimensional space while preserving as much information as possible.
- Find k-dimensional projection, $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(m)}\}$, that ***preserve variance***. ($\boldsymbol{z}^{(i)} \in \mathbb{R}^k$, for i=1,2,…, m)

$$X = \begin{bmatrix} - & \boldsymbol{x}^{(1)^T} & - \\ - & \boldsymbol{x}^{(2)^T} & - \\ - & \boldsymbol{x}^{(3)^T} & - \\ - & \boldsymbol{x}^{(4)^T} & - \\ - & \boldsymbol{x}^{(5)^T} & - \end{bmatrix}_{m \times n}$$

Features

Samples	$x_1^{(i)}$	$x_2^{(i)}$	…	$x_n^{(i)}$
$\boldsymbol{x}^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	…	$x_n^{(1)}$
$\boldsymbol{x}^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	…	$x_n^{(2)}$
⋮	⋮	⋮	⋱	⋮
$\boldsymbol{x}^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	…	$x_n^{(m)}$

University of Mazandaran

PCA Algorithm

Principal Component Analysis(PCA)

Input: Data points in a n-dimensional space, $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ for i=1,2,…, m

Output:

- project into lower dimensional space while preserving as much information as possible.
- Find k-dimensional projection, $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(m)}\}$, that ***preserve variance***. ($\boldsymbol{z}^{(i)} \in \mathbb{R}^k$, for i=1,2,…, m)

Step 0. **(Pre-processing)** (*feature scaling/mean normalization*):

• ***zero mean:*** To mean normalization of the dataset, it is necessary to calculate the mean ($\mu_j := \frac{\sum_{i=1}^{m} x_j^{(i)}}{m}$) and replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$:

$$x_j^{(i)} := x_j^{(i)} - \mu_j \text{ for } j = 1,2,\dots,n.$$

• ***Scaling*** (Optional): If there are different features on different scales, scale feature to have comparable range of values. To this end, compute standard deviation (σ_j) for *new* j^{th} feature and then apply following formula for j^{th} feature of each point in the dataset

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)}}{\sigma_j} \quad where\ {\sigma_j}^2 = \frac{\sum_{i=1}^{m} \left(x_j^{(i)}\right)^2}{m}, \quad j = 1,2,\dots,n.$$

Step 1. Compute covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$.

Step 2. Compute eigenvectors and eigenvalue of Σ.

Step 3. Select top k eigenvectors.

Step 4. Project points onto subspace spanned by the.

University of Mazandaran

PCA Algorithm

Feature scaling

$$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{\sigma_j} = \frac{x_j^{(i)} - \frac{\sum_{i=1}^{m} x_j^{(i)}}{m}}{\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)} - \frac{\sum_{i=1}^{m} x_j^{(i)}}{m}\right)^2}}$$

```
from sklearn.preprocessing import StandardScaler
```

$$x_j^{(i)} := \frac{x_j^{(i)} - \min\limits_{i=1,2,\dots,m} x_j^{(i)}}{\max\limits_{i=1,2,\dots,m} x_j^{(i)} - \min\limits_{i=1,2,\dots,m} x_j^{(i)}} \quad \Rightarrow \quad x_j^{(i)} \in [0, +1]$$

```
from sklearn.preprocessing import MinMaxScaler
```

University of Mazandaran

Covariance

❖ Variance and Covariance:

Measure the "***spread***" of a set of points around their center of mass (mean).

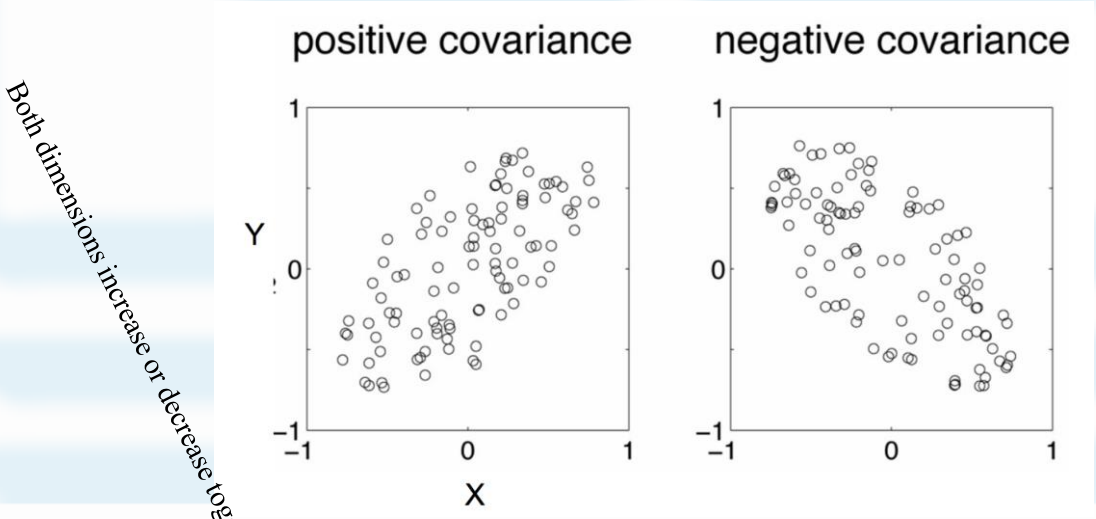
➢ Variance:

Measures the deviation from the mean for points in one dimension.

➢ Covariance:

Measures how much each of the dimensions vary from the mean with **respect to each other.**

- *Covariance is measured between two dimensions.*
- *Covariance sees if there is a relation between two dimensions.*
- *Covariance between one dimension is the variance.*



Both dimensions increase or decrease together

While one increase the other decrease

http://www.cse.psu.edu/~rtc12/CSE586Spring2010/lectures/pcaLectureShort_6pp.pdf

University of Mazandaran

PCA Algorithm

Principal Component Analysis(PCA)

Input: Data points in a n-dimensional space, $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ for i=1,2,..., m

Output:

- project into lower dimensional space while preserving as much information as possible.
- Find k-dimensional projection, $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(m)}\}$, that ***preserve variance***. ($\boldsymbol{z}^{(i)} \in \mathbb{R}^k$, for i=1,2,..., m)

Step 0. **(Pre-processing)** (*feature scaling/mean normalization*)

Step 1. Compute covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$:

$$\Sigma = \frac{1}{m} X^T X = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T \quad s.t. \quad X = \begin{bmatrix} - & \boldsymbol{x}^{(1)^T} & - \\ - & \boldsymbol{x}^{(2)^T} & - \\ & \vdots & \\ - & \boldsymbol{x}^{(m)^T} & - \end{bmatrix}_{m \times n}$$

Step 2. Compute eigenvectors and eigenvalue of Σ:

$$[U, S, V] = svd(\Sigma) \quad s.t \quad \Sigma = USV^T$$

Step 3. Select top k eigenvectors.

$$U = \begin{bmatrix} | & | & \cdots & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & \cdots & | \end{bmatrix}_{n \times n} \longrightarrow U_{reduced} = \begin{bmatrix} | & | & \cdots & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & \cdots & | \end{bmatrix}_{n \times k}$$

Step 4. Project points onto subspace spanned by top k eigenvectors :

$$Z = X \times U_{reduced}$$

$$\boldsymbol{z}^{(i)} = U_{reduced}{}^T \times \boldsymbol{x}^{(i)}, \qquad \text{for } i=1,2,\dots, m$$

University of Mazandaran

PCA Algorithm

Example 1.

	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
$\boldsymbol{x}^{(1)}$	1	2	3	4
$\boldsymbol{x}^{(2)}$	5	5	6	7
$\boldsymbol{x}^{(3)}$	1	4	2	3
$\boldsymbol{x}^{(4)}$	5	3	2	1
$\boldsymbol{x}^{(5)}$	8	1	2	2

Dataset matrix

$x_2^{(3)}$ $\quad i = 3, \quad j = 2$

$\boldsymbol{x}^{(3)} = (1,4,2,3)$

Tian-12-PCA.pdf

University of Mazandaran

PCA Algorithm

Step 0. **(Pre-processing)**

Example 1.

	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
$\boldsymbol{x}^{(1)}$	1	2	3	4
$\boldsymbol{x}^{(2)}$	5	5	6	7
$\boldsymbol{x}^{(3)}$	1	4	2	3
$\boldsymbol{x}^{(4)}$	5	3	2	1
$\boldsymbol{x}^{(5)}$	8	1	2	2

Dataset matrix

	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
$\mu_j =$	4	3	3	3.4
$\sigma_j =$	2.69328	1.41421	1.54919	2.05913

Mean and standard deviation before standardization

Tian-12-PCA.pdf

University of Mazandaran

PCA Algorithm

Step 0. **(Pre-processing)**

Example 1.

Cont.

	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
$\boldsymbol{x}^{(1)}$	1	2	3	4
$\boldsymbol{x}^{(2)}$	5	5	6	7
$\boldsymbol{x}^{(3)}$	1	4	2	3
$\boldsymbol{x}^{(4)}$	5	3	2	1
$\boldsymbol{x}^{(5)}$	8	1	2	2

Dataset matrix

μ_1=4
σ_1 = 3
$x_1^{(5)} = 8$

$x_1^{(5)} \leftarrow \frac{x_1^{(5)} - \mu_1}{\sigma_1}$
= *1.49071*

μ_4=3.4
σ_4 = 2.30217
$x_4^{(1)} = 4$

$x_4^{(1)} \leftarrow \frac{x_4^{(1)} - \mu_4}{\sigma_4}$
= *0.29138*

$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
-1.11803	-0.70711	0	0.29138
0.37268	1.41421	1.93649	1.74831
-1.11803	0.70711	-0.64550	-0.19426
0.37268	0	-0.64550	-1.16554
1.49071	-1.41421	-0.64550	-0.67990

Standardized Dataset

Tian-12-PCA.pdf

University of Mazandaran

PCA Algorithm

Step 0. **(Pre-processing)**

Example 1.

Cont.

	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
$\boldsymbol{x}^{(1)}$	-1.11803	-0.70711	0	0.29138
$\boldsymbol{x}^{(2)}$	0.37268	1.41421	1.93649	1.74831
$\boldsymbol{x}^{(3)}$	-1.11803	0.70711	-0.64550	-0.19426
$\boldsymbol{x}^{(4)}$	0.37268	0	-0.64550	-1.16554
$\boldsymbol{x}^{(5)}$	1.49071	-1.41421	-0.64550	-0.67990

Standardized Dataset

Tian-12-PCA.pdf

University of Mazandaran

PCA Algorithm

Step 0. **(Pre-processing)**

Example 1.

Cont.

	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
$\boldsymbol{x}^{(1)}$	-1.11803	-0.70711	0	0.29138
$\boldsymbol{x}^{(2)}$	0.37268	1.41421	1.93649	1.74831
$\boldsymbol{x}^{(3)}$	-1.11803	0.70711	-0.64550	-0.19426
$\boldsymbol{x}^{(4)}$	0.37268	0	-0.64550	-1.16554
$\boldsymbol{x}^{(5)}$	1.49071	-1.41421	-0.64550	-0.67990

Standardized Dataset

$$X = \begin{bmatrix} - & \boldsymbol{x}^{(1)^T} & - \\ - & \boldsymbol{x}^{(2)^T} & - \\ - & \boldsymbol{x}^{(3)^T} & - \\ - & \boldsymbol{x}^{(4)^T} & - \\ - & \boldsymbol{x}^{(5)^T} & - \end{bmatrix} = \begin{bmatrix} -1.11803 & -0.70711 & 0 & 0.29138 \\ 0.37268 & 1.41421 & 1.93649 & 1.74831 \\ -1.11803 & 0.70711 & -0.64550 & -0.19426 \\ 0.37268 & 0 & -0.64550 & -1.16554 \\ 1.49071 & -1.41421 & -0.64550 & -0.67990 \end{bmatrix}$$

Tian-12-PCA.pdf

University of Mazandaran

PCA Algorithm

Example 1.

Step 1. Compute covariance matrix $\Sigma \in \mathbb{R}^{2\times 2}$

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

$$X = \begin{bmatrix} -1.11803 & -0.70711 & 0 & 0.29138 \\ 0.37268 & 1.41421 & 1.93649 & 1.74831 \\ -1.11803 & 0.70711 & -0.64550 & -0.19426 \\ 0.37268 & 0 & -0.64550 & -1.16554 \\ 1.49071 & -1.41421 & -0.64550 & -0.67990 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1.0000 & -0.3162 & 0.0481 & -0.1810 \\ -0.3162 & 1.0000 & 0.6390 & 0.6181 \\ 0.0481 & 0.6390 & 1.0000 & 0.9404 \\ -0.1810 & 0.6181 & 0.9404 & 1.0000 \end{bmatrix}$$

Tian-12-PCA.pdf

University of Mazandaran

PCA Algorithm

Example 1.

Step 2. Compute eigenvectors and eigenvalue of Σ

$$\Sigma = \begin{bmatrix} 1.0000 & -0.3162 & 0.0481 & -0.1810 \\ -0.3162 & 1.0000 & 0.6390 & 0.6181 \\ 0.0481 & 0.6390 & 1.0000 & 0.9404 \\ -0.1810 & 0.6181 & 0.9404 & 1.0000 \end{bmatrix}$$

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

$$U = \begin{bmatrix} -0.1620 & -0.9171 & 0.3071 & 0.1962 \\ 0.5240 & 0.2069 & 0.8173 & 0.1206 \\ 0.5859 & -0.3205 & -0.1883 & -0.7201 \\ 0.5965 & -0.1159 & -0.4497 & 0.6545 \end{bmatrix}$$

$$S = \begin{bmatrix} 2.5158 & 0 & 0 & 0 \\ 0 & 1.0653 & 0 & 0 \\ 0 & 0 & 0.3939 & 0 \\ 0 & 0 & 0 & 0.0250 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.1620 & -0.9171 & 0.3071 & 0.1962 \\ 0.5240 & 0.2069 & 0.8173 & 0.1206 \\ 0.5859 & -0.3205 & -0.1883 & -0.7201 \\ 0.5965 & -0.1159 & -0.4497 & 0.6545 \end{bmatrix}$$

$\lambda_1 = 2.5158$ $e_1 = [-0.1620 \quad 0.5240 \quad 0.5859 \quad 0.5965]^T$ — The first principal component

$\lambda_2 = 1.0653$ $e_2 = [-0.9171 \quad 0.2069 \quad -0.3205 \quad -0.1159]^T$ — The Second principal component

$\lambda_3 = 0.3939$ $e_3 = [0.3071 \quad 0.8173 \quad -0.1883 \quad -0.4497]^T$ — The third principal component

$\lambda_4 = 0.0250$ $e_4 = [0.1962 \quad 0.1206 \quad -0.7201 \quad 0.6545]^T$ — The fourth principal component

University of Mazandaran

PCA Algorithm

Example 1.

Step 3. Select top k eigenvectors, k=2

$$U_{reduced} \coloneqq U(:, 1{:}2) = \begin{bmatrix} -0.1620 & -0.9171 \\ 0.5240 & 0.2069 \\ 0.5859 & -0.3205 \\ 0.5965 & -0.1159 \end{bmatrix}$$

Step 4. Project points onto subspace spanned by top k=2 eigenvectors:

$$Z = \begin{bmatrix} - & \boldsymbol{z}^{(1)^T} & - \\ - & \boldsymbol{z}^{(2)^T} & - \\ - & \boldsymbol{z}^{(3)^T} & - \\ - & \boldsymbol{z}^{(4)^T} & - \\ - & \boldsymbol{z}^{(5)^T} & - \end{bmatrix} = X \times U_{reduced} = \begin{bmatrix} -0.0157 & 0.8452 \\ 2.8583 & -0.8725 \\ 0.0576 & 1.4010 \\ -1.1339 & 0.0003 \\ -1.7663 & -1.3740 \end{bmatrix}$$

$$\boldsymbol{z}^{(i)} = {U_{reduced}}^T \times \boldsymbol{x}^{(i)},$$

for i=1,2,..., m

	$\boldsymbol{z}_1^{(i)}$	$\boldsymbol{z}_2^{(i)}$
$\boldsymbol{z}^{(1)}$	−0.0157	0.8452
$\boldsymbol{z}^{(2)}$	2.8583	−0.8725
$\boldsymbol{z}^{(3)}$	0.0576	1.4010
$\boldsymbol{z}^{(4)}$	-1.1339	0.0003
$\boldsymbol{z}^{(5)}$	-1.7663	-1.3740

University of Mazandaran

PCA Algorithm

Example 1.

- ***Reconstruction from compressed representation***, k=2

$$\boldsymbol{Z} = \begin{bmatrix} -0.0157 & 0.8452 \\ 2.8583 & -0.8725 \\ 0.0576 & 1.4010 \\ -1.1339 & 0.0003 \\ -1.7663 & -1.3740 \end{bmatrix}$$

$$\boldsymbol{X_{approx}}^{(i)} = U_{reduced} \times \boldsymbol{z}^{(i)}, \quad \text{for } i=1,2,\ldots, m$$

$$\boldsymbol{X_{approx}} = \boldsymbol{Z} \times {U_{reduced}}^{T}$$

$$\boldsymbol{X_{approx}} = \begin{bmatrix} -0.7726 & 0.1667 & -0.2801 & -0.1073 \\ 0.3373 & 1.3173 & 1.9543 & 1.8063 \\ -1.2942 & 0.3201 & -0.4154 & -0.1281 \\ 0.1834 & -0.5941 & -0.6644 & -0.6764 \\ 1.5461 & -1.2100 & -0.5945 & -0.8944 \end{bmatrix}$$

University of Mazandaran

PCA Algorithm

Example 2.

	$x_1^{(i)}$	$x_2^{(i)}$
$\boldsymbol{x}^{(1)}$	2.5	2.4
$\boldsymbol{x}^{(2)}$	0.5	0.7
$\boldsymbol{x}^{(3)}$	2.2	2.9
$\boldsymbol{x}^{(4)}$	1.9	2.2
$\boldsymbol{x}^{(5)}$	3.1	3.0
$\boldsymbol{x}^{(6)}$	2.3	2.7
$\boldsymbol{x}^{(7)}$	2	1.6
$\boldsymbol{x}^{(8)}$	1	1.1
$\boldsymbol{x}^{(9)}$	1.5	1.6
$\boldsymbol{x}^{(10)}$	1.1	0.9

Original Dataset

	$x_1^{(i)}$	$x_2^{(i)}$
$\mu_j =$	1.81	1.91

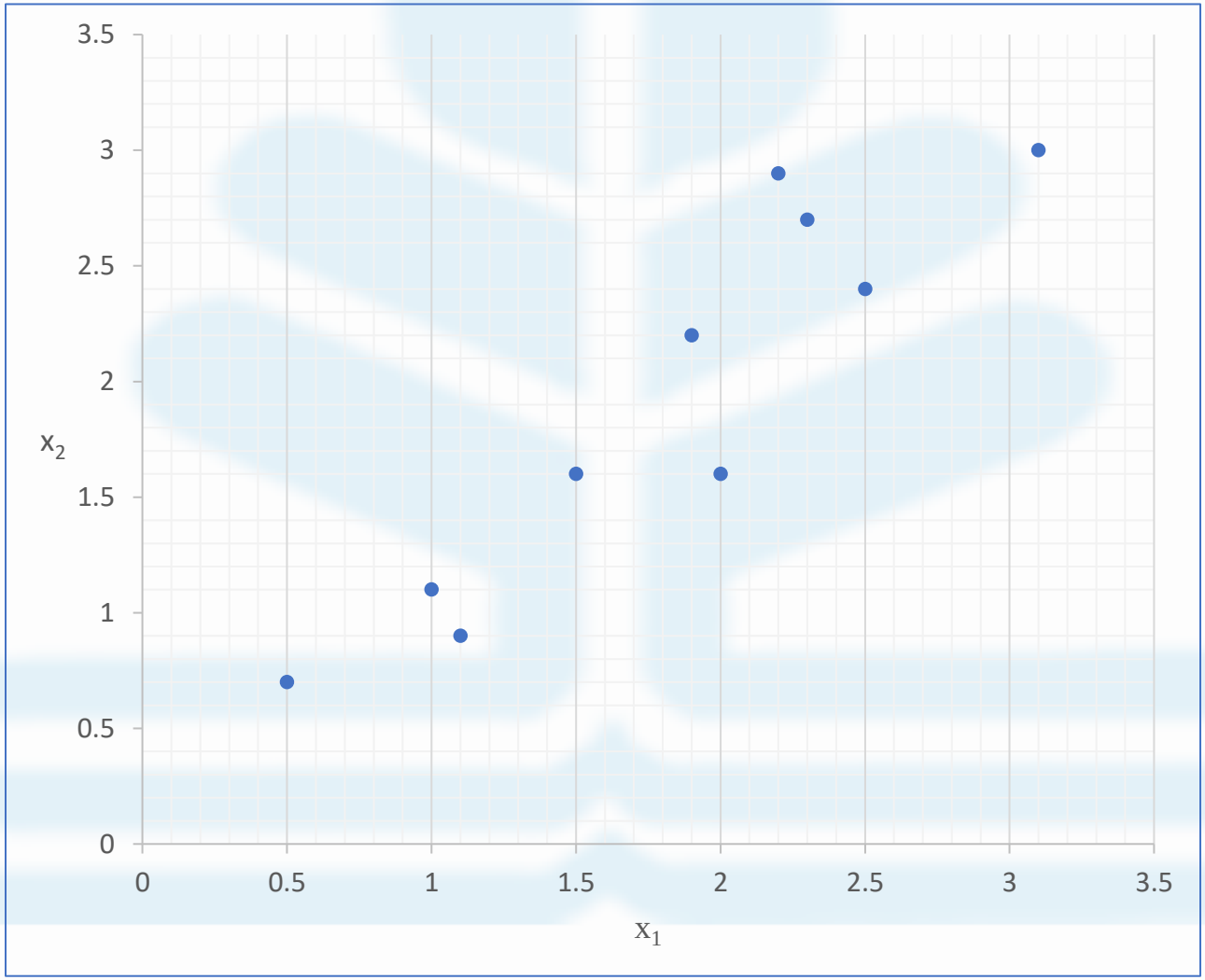
University of Mazandaran

PCA Algorithm

Example 2.

Cont.

Dataset



University of Mazandaran

PCA Algorithm

$$x_j^{(i)} := x_j^{(i)} - \mu_j$$

Example 2.

Step 0. **(Pre-processing)** ***zero-mean*****:**

	$x_1^{(i)}$	$x_2^{(i)}$
$\boldsymbol{x}^{(1)}$	0.69	0.49
$\boldsymbol{x}^{(2)}$	-1.31	-1.21
$\boldsymbol{x}^{(3)}$	.39	0.99
$\boldsymbol{x}^{(4)}$	0.09	0.29
$\boldsymbol{x}^{(5)}$	1.29	1.09
$\boldsymbol{x}^{(6)}$	0.49	0.79
$\boldsymbol{x}^{(7)}$	0.19	-0.31
$\boldsymbol{x}^{(8)}$	-0.81	-0.81
$\boldsymbol{x}^{(9)}$	-0.31	-0.31
$\boldsymbol{x}^{(10)}$	-0.71	-1.01

Mean Adjusted Dataset

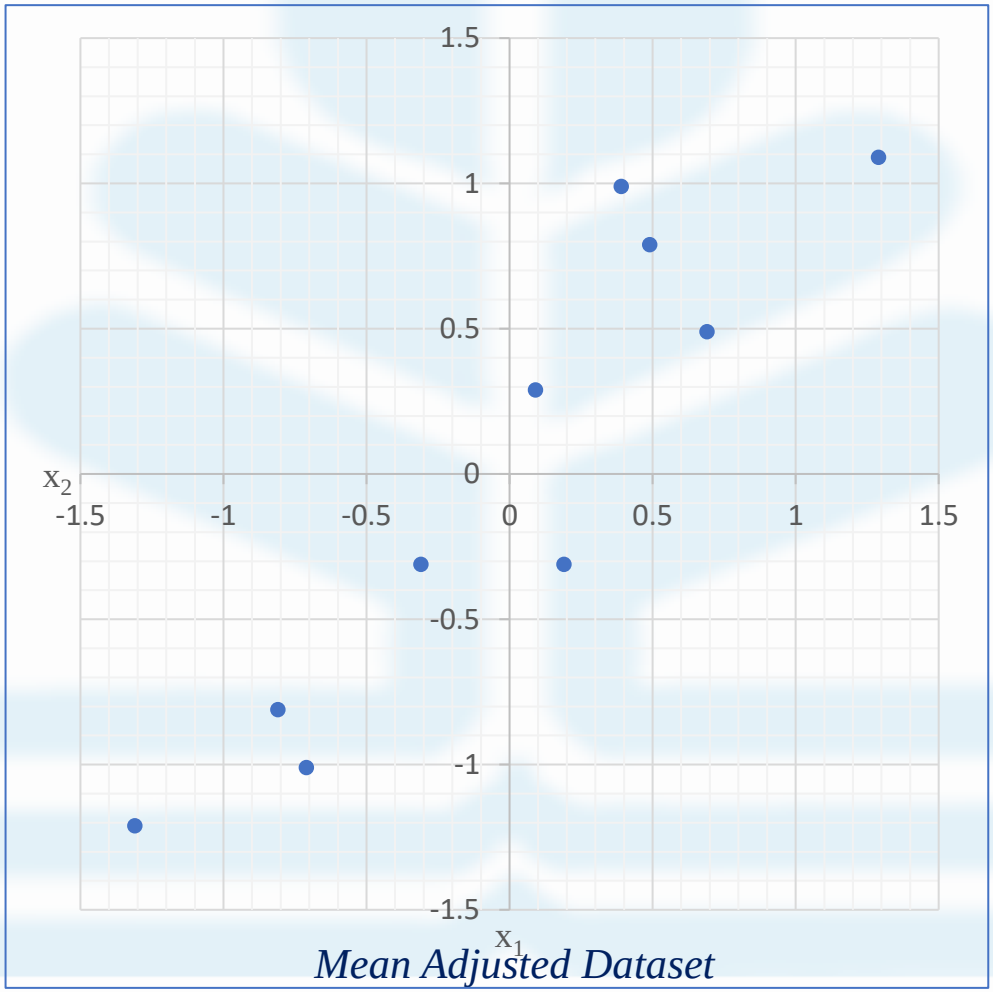
University of Mazandaran

PCA Algorithm

Cont.

Example 2.

Step 0. **(Pre-processing)** ***zero-mean:***



Mean Adjusted Dataset

University of Mazandaran

PCA Algorithm

Example 2.

Step 1. Compute covariance matrix $\Sigma \in \mathbb{R}^{2\times 2}$

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

$$X = \begin{bmatrix} 0.69 & 0.49 \\ -1.31 & -1.21 \\ 0.39 & 0.99 \\ 0.09 & 0.29 \\ 1.29 & 1.09 \\ 0.49 & 0.79 \\ 0.19 & -0.31 \\ -0.81 & -0.81 \\ -0.31 & -0.31 \\ -0.71 & -1.01 \end{bmatrix} \quad \Rightarrow \quad \Sigma = \begin{bmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555556 \end{bmatrix}$$

University of Mazandaran

PCA Algorithm

Example 2.

Step 2. Compute eigenvectors and eigenvalue of Σ

$$\Sigma = \begin{bmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555556 \end{bmatrix}$$

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

$$U = \begin{bmatrix} -0.677873399 & -0.735178656 \\ -0.735178656 & 0.677873399 \end{bmatrix}$$

$$S = \begin{bmatrix} 1.28402771 & 0 \\ 0 & 0.049083399 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.677873399 & -0.735178656 \\ -0.735178656 & 0.677873399 \end{bmatrix}$$

$$\lambda_1 = 1.28402771 \qquad e_1 = \begin{bmatrix} -0.677873399 \\ -0.735178656 \end{bmatrix}$$

The eigenvector with the highest eigenvalue is the first principal component of the data set

$$\lambda_2 = 0.049083399 \qquad e_2 = \begin{bmatrix} -0.735178656 \\ 0.677873399 \end{bmatrix}$$

The eigenvector with the second highest eigenvalue is the second principal component of the data set

University of Mazandaran

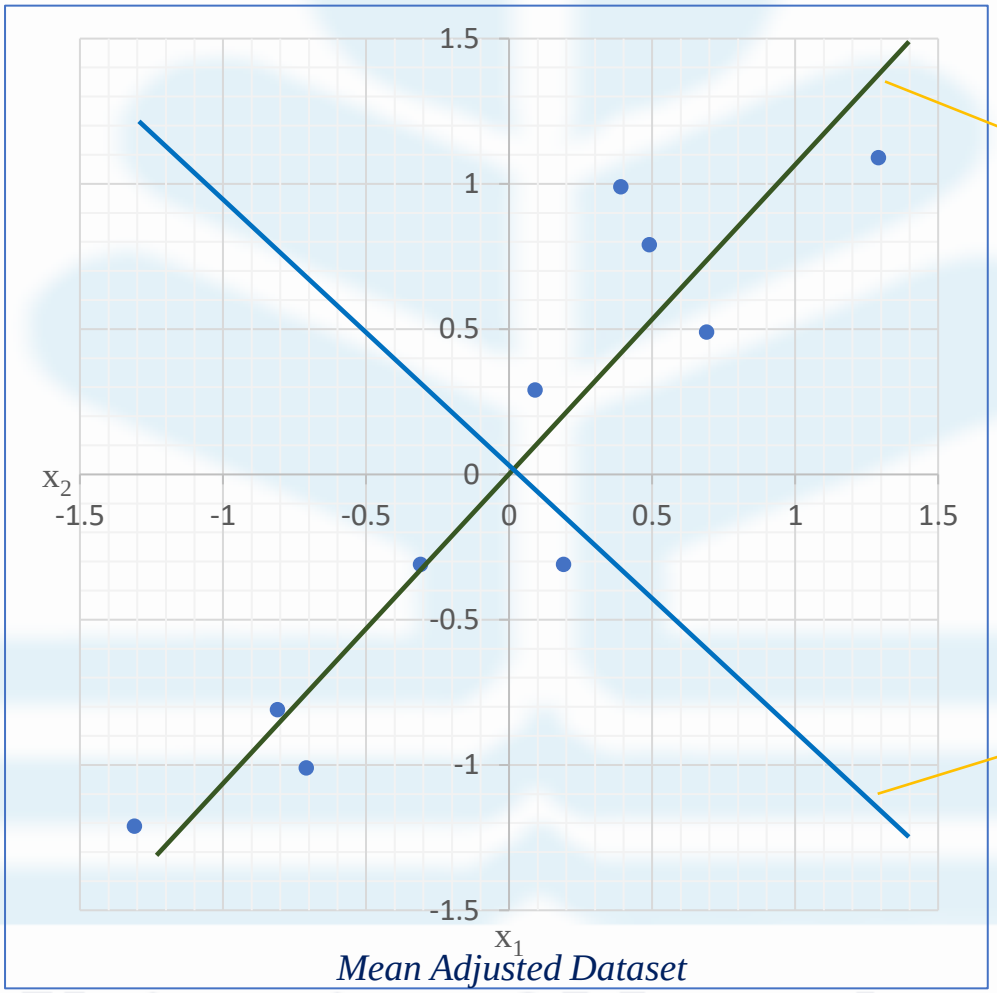
PCA Algorithm

Cont.

Example 2.

Step 2. Compute eigenvectors and eigenvalue of Σ

Mean Adjusted Dataset and eigenvectors of Σ



Mean Adjusted Dataset

The first principle component

The Second principle component

University of Mazandaran

PCA Algorithm

Example 2.

Step 3. Select top k eigenvectors, k=2

$$U_{reduced} := U = \begin{bmatrix} -0.677873399 & -0.735178656 \\ -0.735178656 & 0.677873399 \end{bmatrix}$$

Step 4. Project points onto subspace spanned by top k=2 eigenvectors:

$$Z = X \times U_{reduced} = \begin{bmatrix} -0.8280 & -0.1751 \\ 1.7776 & 0.1429 \\ -0.9922 & 0.3844 \\ -0.2742 & 0.1304 \\ -1.6758 & -0.2095 \\ -0.9129 & 0.1753 \\ 0.0991 & -0.3498 \\ 1.1446 & 0.0464 \\ 0.4380 & 0.0178 \\ 1.2238 & -0.1627 \end{bmatrix}$$

$$\boldsymbol{z}^{(i)} = {U_{reduced}}^T \times \boldsymbol{x}^{(i)},$$

for i=1,2,..., m

	$\boldsymbol{z}_1^{(i)}$	$\boldsymbol{z}_2^{(i)}$
$\boldsymbol{z}^{(1)}$	−0.8280	−0.1751
$\boldsymbol{z}^{(2)}$	1.7776	0.1429
$\boldsymbol{z}^{(3)}$	-0.9922	0.3844
$\boldsymbol{z}^{(4)}$	-0.2742	0.1304
$\boldsymbol{z}^{(5)}$	-1.6758	-0.2095
$\boldsymbol{z}^{(6)}$	-0.9129	0.1753
$\boldsymbol{z}^{(7)}$	0.0991	-0.3498
$\boldsymbol{z}^{(8)}$	0.0991	0.0464
$\boldsymbol{z}^{(9)}$	0.4380	0.0178
$\boldsymbol{z}^{(10)}$	1.2238	-0.1627

University of Mazandaran

PCA Algorithm

Cont.

Example 2.

Step 3. Select top k eigenvectors, k=1

$$U_{reduced} := U = \begin{bmatrix} -0.677873399 \\ -0.735178656 \end{bmatrix}$$

Step 4. Project points onto subspace spanned by top k=1 eigenvector:

$$Z = X \times U_{reduced} = \begin{bmatrix} -0.8280 \\ 1.7776 \\ -0.9922 \\ -0.2742 \\ -1.6758 \\ -0.9129 \\ 0.0991 \\ 1.1446 \\ 0.4380 \\ 1.2238 \end{bmatrix}$$

	$z_1^{(i)}$
$z^{(1)}$	−0.8280
$z^{(2)}$	1.7776
$z^{(3)}$	-0.9922
$z^{(4)}$	-0.2742
$z^{(5)}$	-1.6758
$z^{(6)}$	-0.9129
$z^{(7)}$	0.0991
$z^{(8)}$	0.0991
$z^{(9)}$	0.4380
$z^{(10)}$	1.2238

$$z^{(i)} = {U_{reduced}}^T \times x^{(i)},$$

for i=1,2,..., m

University of Mazandaran

PCA Algorithm

Example 2.

- ***Reconstruction from compressed representation***, k=1

$$z^{(1)} = \begin{bmatrix} -0.8280 \\ 1.7776 \\ -0.9922 \\ -0.2742 \\ -1.6758 \\ -0.9129 \\ 0.0991 \\ 1.1446 \\ 0.4380 \\ 1.2238 \end{bmatrix}$$

$$x_{approx}^{(i)} = U_{reduced} \times z^{(i)}, \quad \text{for } i=1,2,\ldots, m$$

$$x_{approx}^{(1)} = \begin{bmatrix} 0.5613 & 0.6087 \\ -1.2050 & -1.3068 \\ 0.6726 & 0.7294 \\ 0.1859 & 0.2016 \\ 1.1360 & 1.2320 \\ 0.6189 & 0.6712 \\ -0.0672 & -0.0729 \\ -0.7759 & -0.8415 \\ -0.2969 & -0.3220 \\ -0.8296 & -0.8997 \end{bmatrix}$$

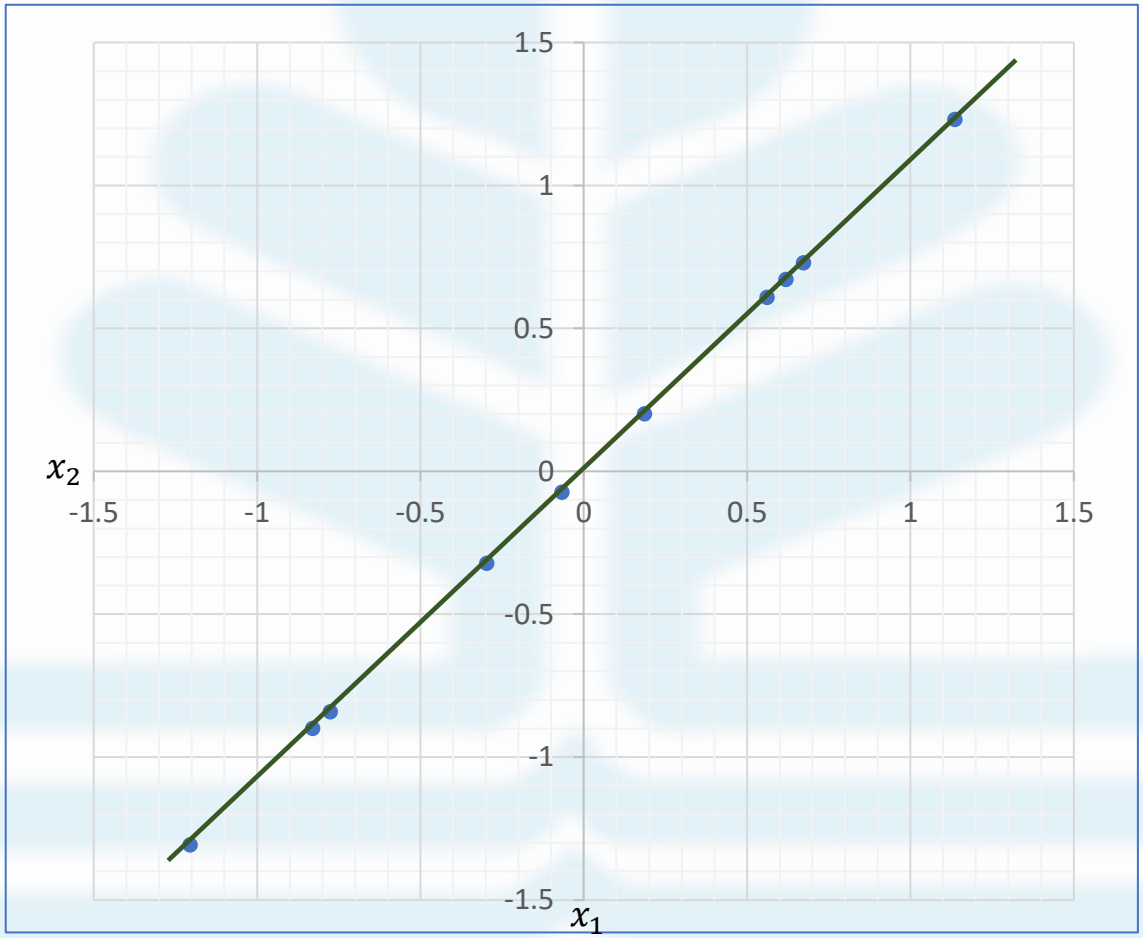
University of Mazandaran

Reconstruction from compressed representation

Cont.

Example 2.

- ***Reconstruction from compressed representation***, k=1



University of Mazandaran

PCA Python Code

```
import numpy as np
import pandas as pd

A=np.matrix([[ 2.5, 2.4],
             [ 0.5, 0.7],
             [ 2.2, 2.9],
             [ 1.9, 2.2],
             [ 3.1, 3.0],
             [ 2.3, 2.7],
             [ 2.0, 1.6],
             [ 1.0, 1.1],
             [ 1.5, 1.6],
             [ 1.1, 0.9]])

df = pd.DataFrame(A,columns  = ['f1','f2'])
```

	f1	f2
0	0.69	0.49
1	-1.31	-1.21
2	0.39	0.99
3	0.09	0.29
4	1.29	1.09
5	0.49	0.79
6	0.19	-0.31
7	-0.81	-0.81
8	-0.31	-0.31
9	-0.71	-1.01

```
df_std  = (df - df.mean())
#df_std  = (df - df.mean()) / (df.std())
print(df_std)

#Covariance population formula (divide by N)
df_cov = np.cov(df_std.T, bias = 1)

#Covariance sample formula (divide by N-1)
cov_mat = np.cov(df_std.T, bias = 0)
```

University of Mazandaran

PCA Python Code

```
eigen_val, eigen_vectors = np.linalg.eig(cov_mat)

print('\neigen_val:\n',eigen_val)
print('\neigen_vectors:\n',eigen_vectors)

descending_indices=np.argsort(eigen_val)[::-1]
eigen_val=eigen_val[descending_indices]
eigen_vectors=eigen_vectors[:,descending_indices]
```

```
eigen_val:
 [0.0490834  1.28402771]

eigen_vectors:
 [[-0.73517866 -0.6778734 ]
 [ 0.6778734  -0.73517866]]
```

University of Mazandaran

PCA Python Code

```
n_components=1
U_reduced = eigen_vectors[:,:n_components]
transformed_data = np.matmul(np.array(df_std),U_reduced)

pd.DataFrame(data = transformed_data
   ,columns = ['principal component '+ str(i+1) for i in range(n_components)])
```

	principal component 1
0	-0.827970
1	1.777580
2	-0.992197
3	-0.274210
4	-1.675801
5	-0.912949
6	0.099109
7	1.144572
8	0.438046
9	1.223821

University of Mazandaran

PCA Python Code

```
df2=x_approx=np.matmul(transformed_data,U_reduced.T) +np.mean(A)
pd.DataFrame(data = x_approx
             , columns = ['x '+ str(i+1) for i in range(n)])
```

	x 1	x 2
0	2.421259	2.468706
1	0.655026	0.553161
2	2.532584	2.589442
3	2.045880	2.061594
4	2.995981	3.092013
5	2.478864	2.531181
6	1.792816	1.787137
7	1.084125	1.018535
8	1.563060	1.537958
9	1.030405	0.960273

University of Mazandaran

PCA using sklearn

```
import numpy as np
import pandas as pd

A=np.matrix([[ 2.5, 2.4],
             [ 0.5, 0.7],
             [ 2.2, 2.9],
             [ 1.9, 2.2],
             [ 3.1, 3.0],
             [ 2.3, 2.7],
             [ 2.0, 1.6],
             [ 1.0, 1.1],
             [ 1.5, 1.6],
             [ 1.1, 0.9]])
df = pd.DataFrame(A,columns  = ['f1','f2'])

df_std  = (df - df.mean())
#df_std  = (df - df.mean()) / (df.std())

n_components=1
```

	principal component 1
0	-0.827970
1	1.777580
2	-0.992197
3	-0.274210
4	-1.675801
5	-0.912949
6	0.099109
7	1.144572
8	0.438046
9	1.223821

```
from sklearn.decomposition import PCA
pca = PCA(n_components=n_components)
principalComponents = pca.fit_transform(df_std)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component '+ str(i+1) for i in range(n_components)])

principalDf
```

University of Mazandaran

PCA using sklearn

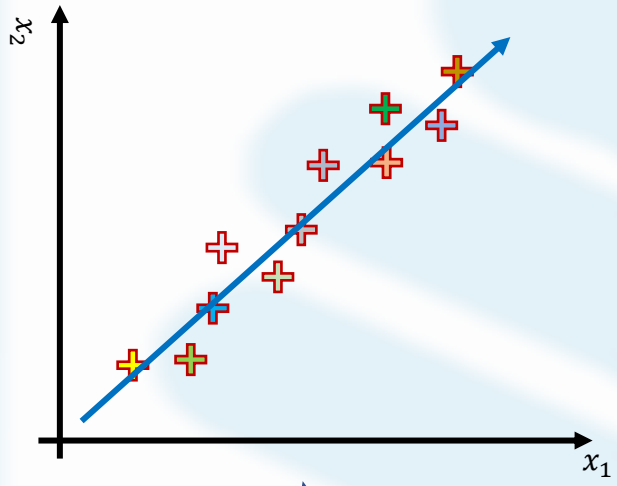
```
from sklearn.decomposition import KernelPCA
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd

A=np.matrix([[ 2.5, 2.4],
             [ 0.5, 0.7],
             [ 2.2, 2.9],
             [ 1.9, 2.2],
             [ 3.1, 3.0],
             [ 2.3, 2.7],
             [ 2.0, 1.6],
             [ 1.0, 1.1],
             [ 1.5, 1.6],
             [ 1.1, 0.9]])

n_components=1
scaler = StandardScaler()
scaler.fit(A)
transformed_A=scaler.transform(A)

transformer = KernelPCA(n_components, kernel='linear')
X_transformed = transformer.fit_transform(transformed_A)

principalDf = pd.DataFrame(data = X_transformed
             , columns = ['principal component '+ str(i+1) for i in range(n_components)])
principalDf
```

	principal component 1
0	-1.086432
1	2.308937
2	-1.241919
3	-0.340782
4	-2.184290
5	-1.160739
6	0.092605
7	1.482108
8	0.567226
9	1.563287

University of Mazandaran

Reconstruction from compressed representation

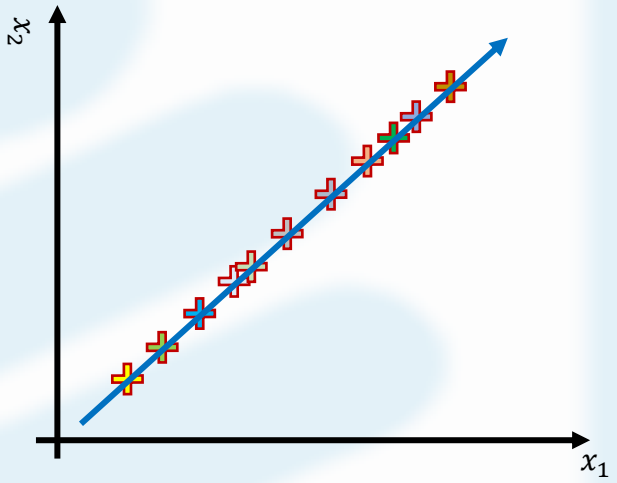
Reduce data from 2D to 1D



1

$$z^{(i)} = U_{reduced}{}^{T} \times x^{(i)},$$
for i=1,2,..., m

2

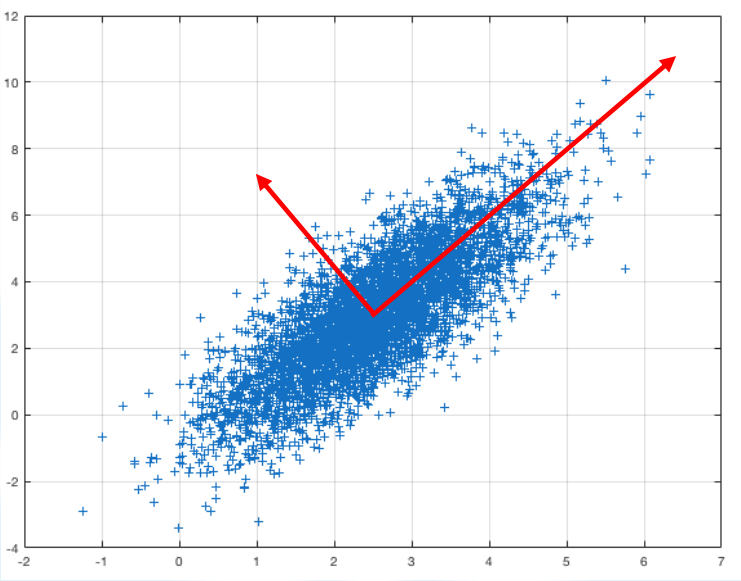


$$x_{approx}{}^{(i)} = U_{reduced} \times z^{(i)},$$
for i=1,2,..., m

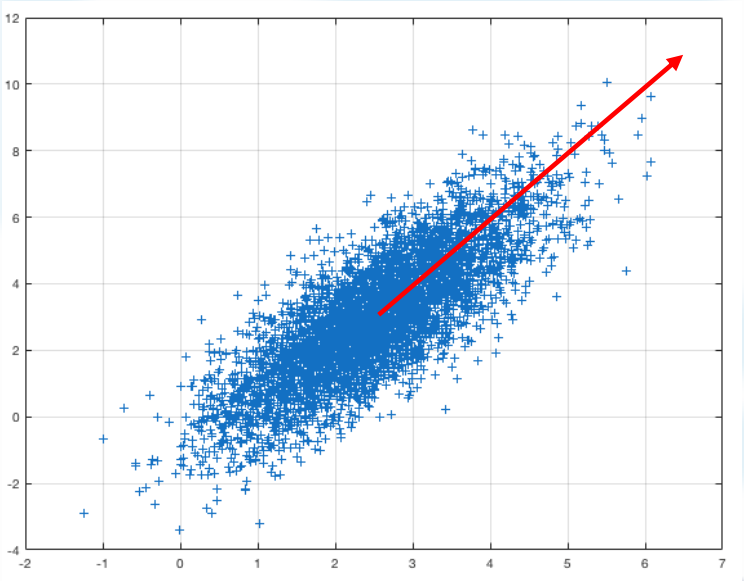
https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction

University of Mazandaran

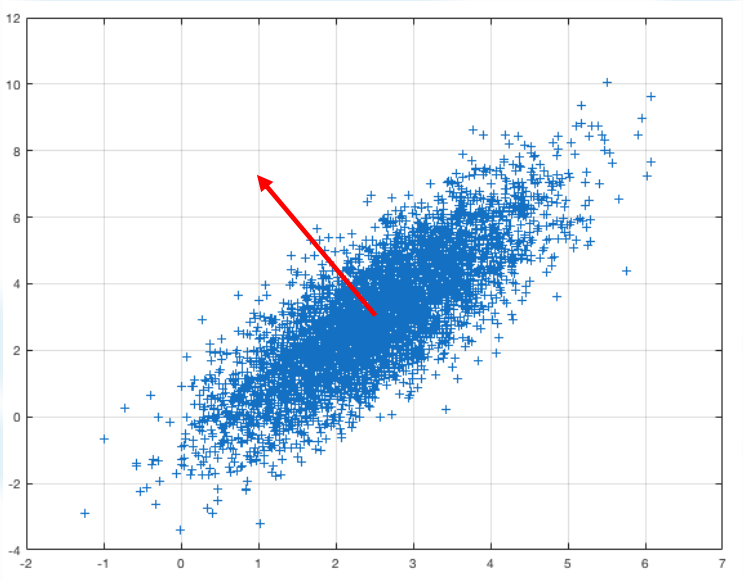
PCA



1st PCA axis



2nd PCA axis



University of Mazandaran

Choosing k (number of principal components)

Typically, for a constant value $\epsilon < 1$, choose k to be smallest value so that:

$$R(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{approx}}, k) := \frac{\frac{1}{m}\sum_{i=1}^{m}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}_{\boldsymbol{approx}}{}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|\boldsymbol{x}^{(i)}\right\|^2} \leq \epsilon$$

Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}_{\boldsymbol{approx}}{}^{(i)}\right\|^2$

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

$$If\ R(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{approx}}, k) \leq \epsilon, then$$
$$100(1-\epsilon)\ precent\ of\ variance\ is\ retained.$$

e. g. If $R(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{approx}}, k) \leq 0.01$, then 99% of variance is retained.
If $R(\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{approx}}, k) \leq 0.05$, then 95% of variance is retained.

University of Mazandaran

Choosing k (number of principal components)

Typically, for a constant value $p \leq 100$, choose k to be smallest value so that:

$$R(S,k) := \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq p,$$

where

$$[U,S,V] = svd(\Sigma) \quad \text{given} \quad \Sigma = \frac{1}{m} X^T X = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)}\right)\left(x^{(i)}\right)^T$$

$$If\ R(S,k) \geq p, then\ p percent\ of\ variance\ is\ retained.$$

e. g. If $R(S,k) \geq 99$, then 99% of variance is retained.
If $R(S,k) \geq 95$, then 95% of variance is retained.

University of Mazandaran

Kernel PCA

Lemma: *If* u *is an eigenvector of the matrix* Σ *then* u *is a linear combination of the samples* $x^{(i)}$ *for* $i = 1,2,\dots,m$.

$$\Sigma u = \lambda u \quad \Longrightarrow \quad u = \sum_{i=1}^{m} \alpha_i x^{(i)}$$

Lemma: *To calculate* $\boldsymbol{\alpha} \in \mathbb{R}^m$ *just use Gram matrix* $K = (k_{ij})$, *where* $K_{ij} = {x^{(i)}}^T x^{(i)}$ *is inner product. Don't need the actual value of* $x^{(i)}$.

$K\boldsymbol{\alpha} = m\lambda\boldsymbol{\alpha}$, if K is invertible

https://www.cs.cmu.edu/~bapoczos/Classes/ML10715_2015Fall/slides/PCA.pdf

University of Mazandaran

Kernel PCA

We use α to *calculate the projection* of a new sample *t*:

$$u^T t = \left(\sum_{i=1}^{m} \alpha_i x^{(i)}\right)^T t = \sum_{i=1}^{m} \alpha_i x^{(i)} K\left(x^{(i)}, t\right)$$

Again Don't need the actual value of $x^{(i)}$.

$$K_{ij} = \left\langle \phi\left(x^{(i)}\right), \phi\left(x^{(j)}\right) \right\rangle$$

The data should be centered in the feature space, too! But this is manageable.

$$\tilde{K}_{ij} \doteq \left\langle \phi\left(x^{(i)}\right) - \sum_{k=1}^{m} \phi\left(x^{(k)}\right), \phi\left(x^{(j)}\right) - \sum_{k=1}^{m} \phi\left(x^{(k)}\right) \right\rangle$$

```
from sklearn.datasets import load_digits
from sklearn.decomposition import KernelPCA
X, _ = load_digits(return_X_y=True)
transformer = KernelPCA(n_components=7, kernel='linear')
X_transformed = transformer.fit_transform(X)
```

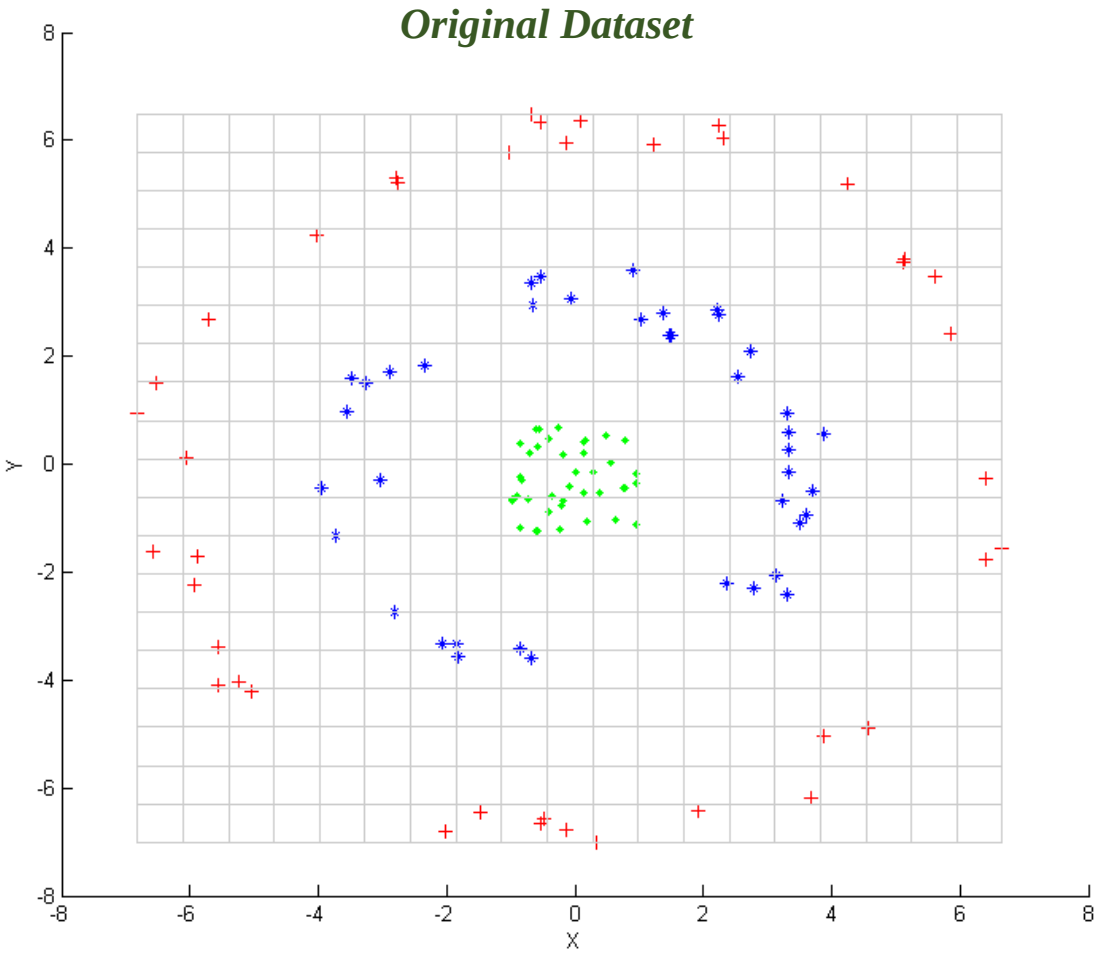
https://www.cs.cmu.edu/~bapoczos/Classes/ML10715_2015Fall/slides/PCA.pdf

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html

University of Mazandaran

Input points before kernel PCA

Example 3.



https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

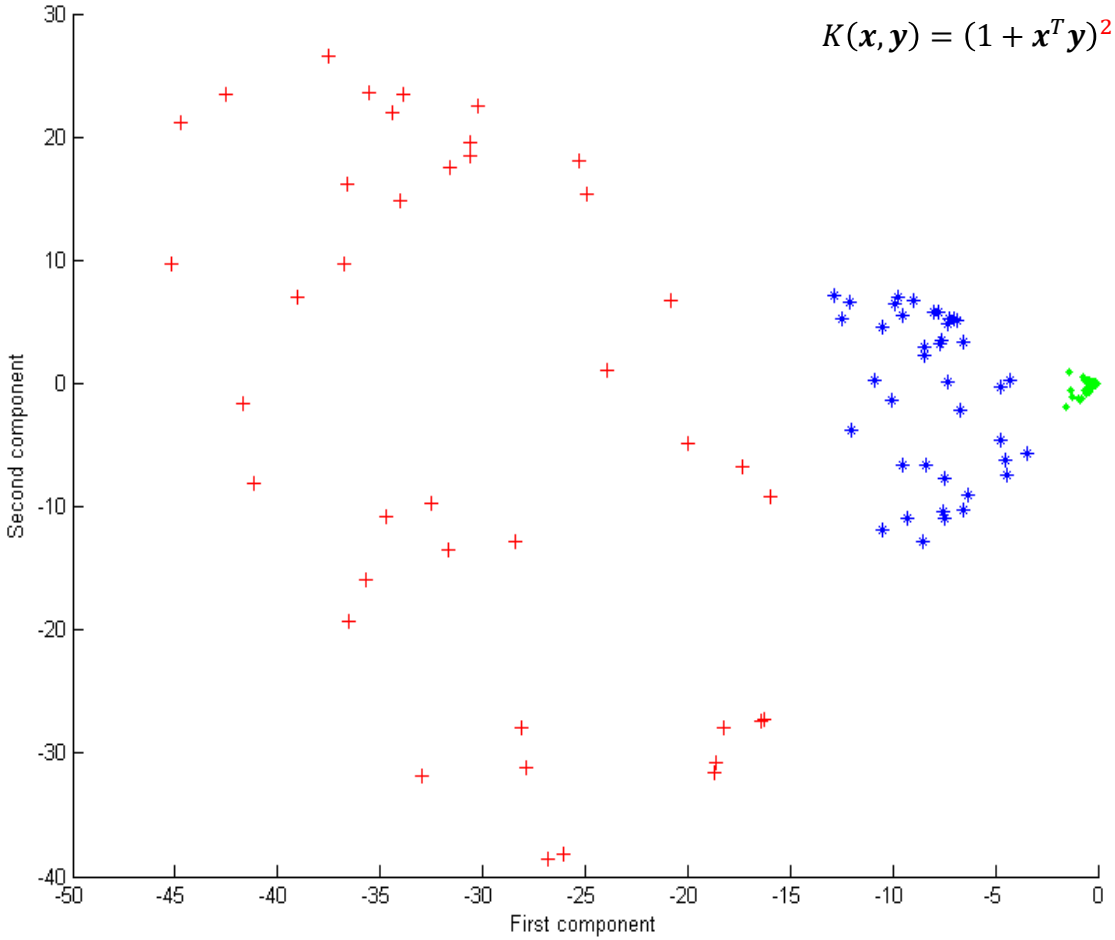
University of Mazandaran

Output after kernel PCA

Example 3.

Polynomial kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = (1 + \boldsymbol{x}^T \boldsymbol{y})^2$$



The three groups are distinguishable using the first component only

https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

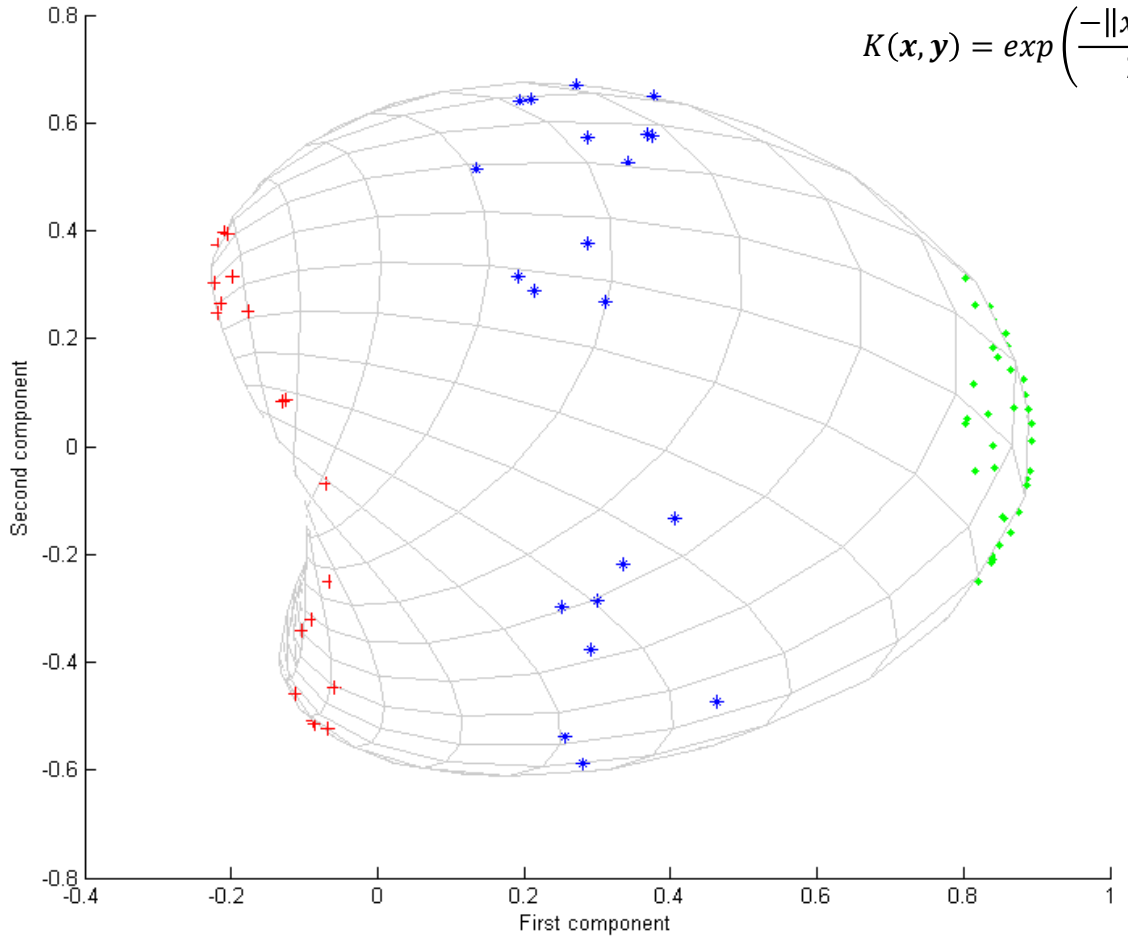
University of Mazandaran

Kernel PCA

Example 3.

Gaussian kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = exp\left(\frac{-\|x-y\|^2}{2\sigma^2}\right)$$



The three groups are distinguishable using the first component only

https://en.wikipedia.org/wiki/Kernel_principal_component_analysis
https://www.cs.cmu.edu/~bapoczos/Classes/ML10715_2015Fall/slides/PCA.pdf

University of Mazandaran

References

[1] https://share.coursera.org/wiki/index.php/ML:Dimensionality_Reduction.
[2] https://www.geeksforgeeks.org/mathematical-approach-to-pca/
[3] https://github.com/gskdhiman/Understanding-PCA/blob/master/PCA_code.ipynb
[4] https://www.cs.cmu.edu/~bapoczos/Classes/ML10715_2015Fall/slides/PCA.pdf
[5] https://en.wikipedia.org/wiki/Kernel_principal_component_analysis
[6] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html
[7] https://www.google.com/search?client=firefox-b-d&q=Tian-12-PCA.pdf
[8] Dimensionality Reduction, CSEP 546, Fereshteh Sadeghi

University of Mazandaran