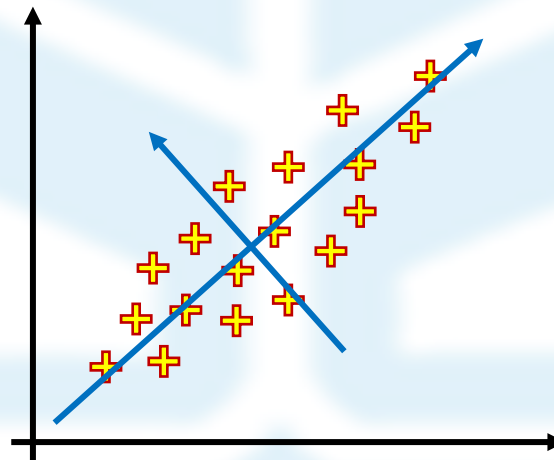


Machine Learning

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**

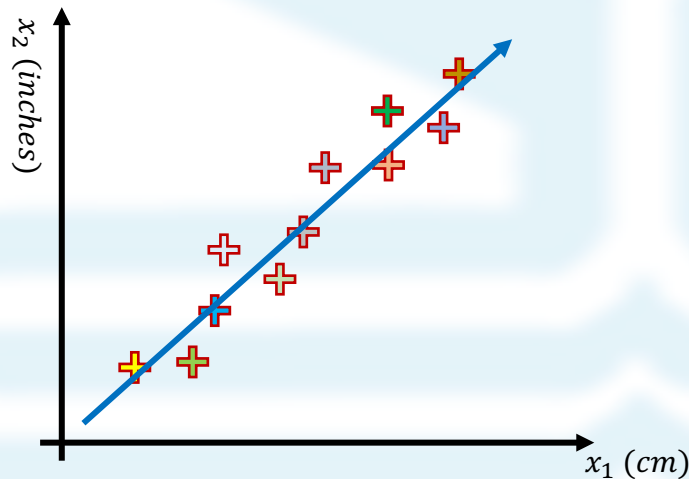
Motivation

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



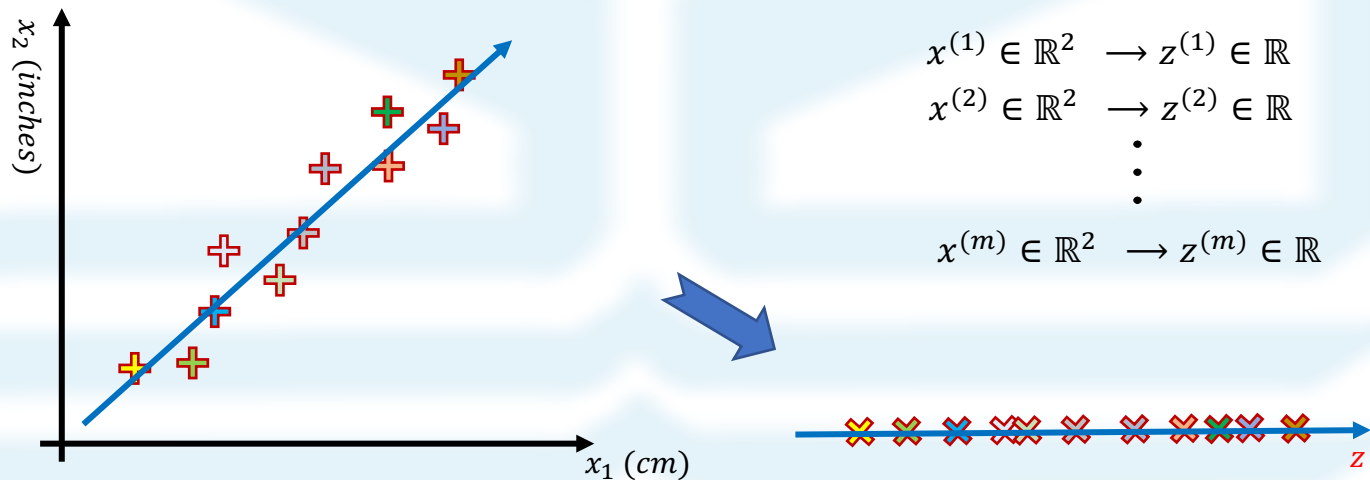
Motivation

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



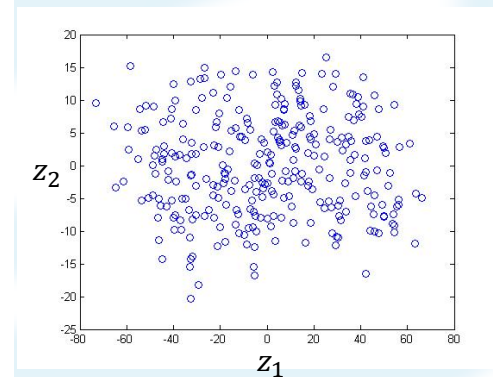
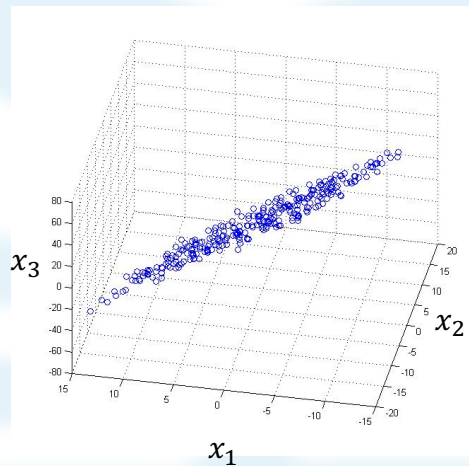
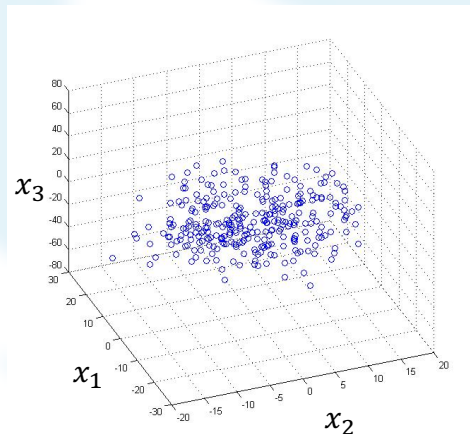
Motivation

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



Motivation

Data Visualization:

- How can we visualize the measurements?

Country	GDP (trillions of US\$)	Per capita GDP (thousands of intl. \$)	Human Development 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	...
...	...	...	...	...	...	...	...

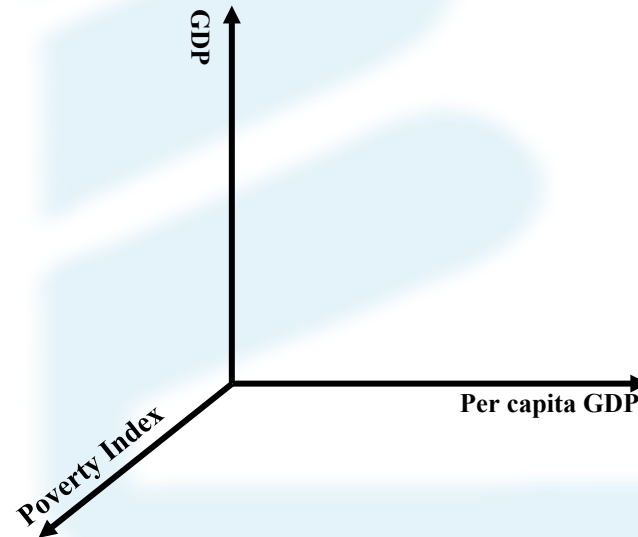
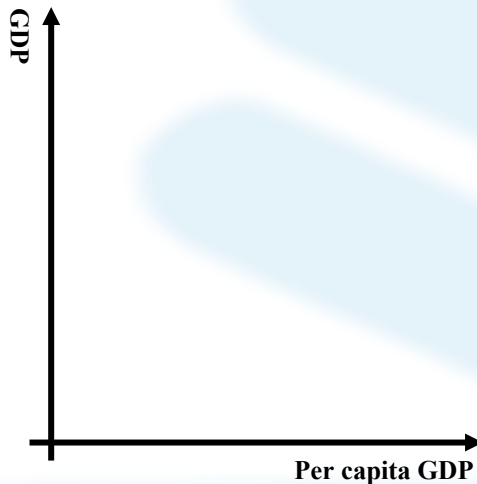
❖ When there are more than three features, it is difficult to see the correlations between the features.

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.

Motivation

Data Visualization:

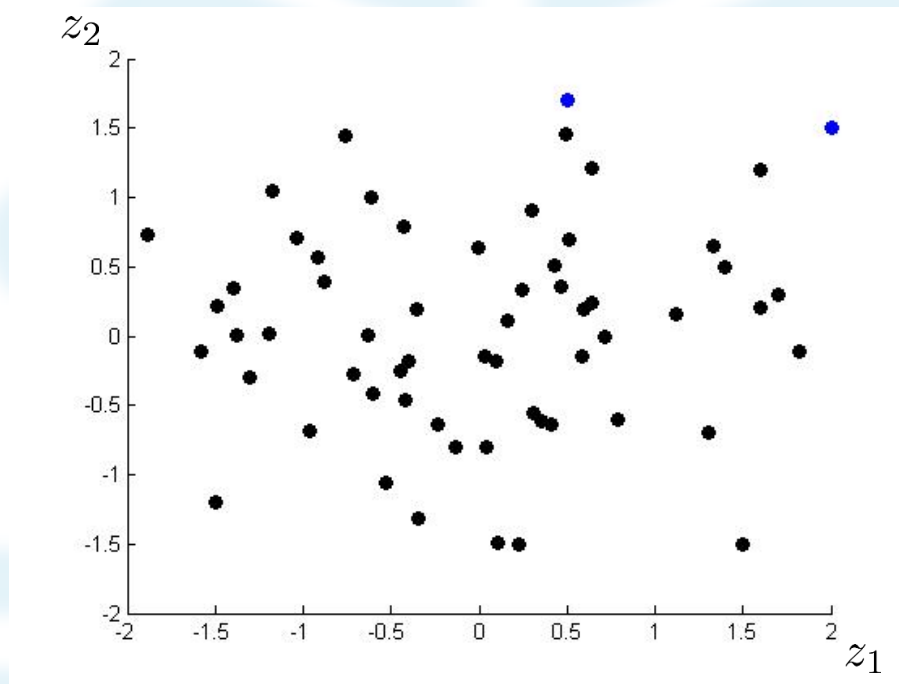
- It is not easy to visualize data that is more than three dimensions. We can reduce the dimensions of our data to three or less in order to plot it. To do this, We need to *find new features*, (z_1 , z_2 and perhaps z_3) that can effectively summarize all the other features.

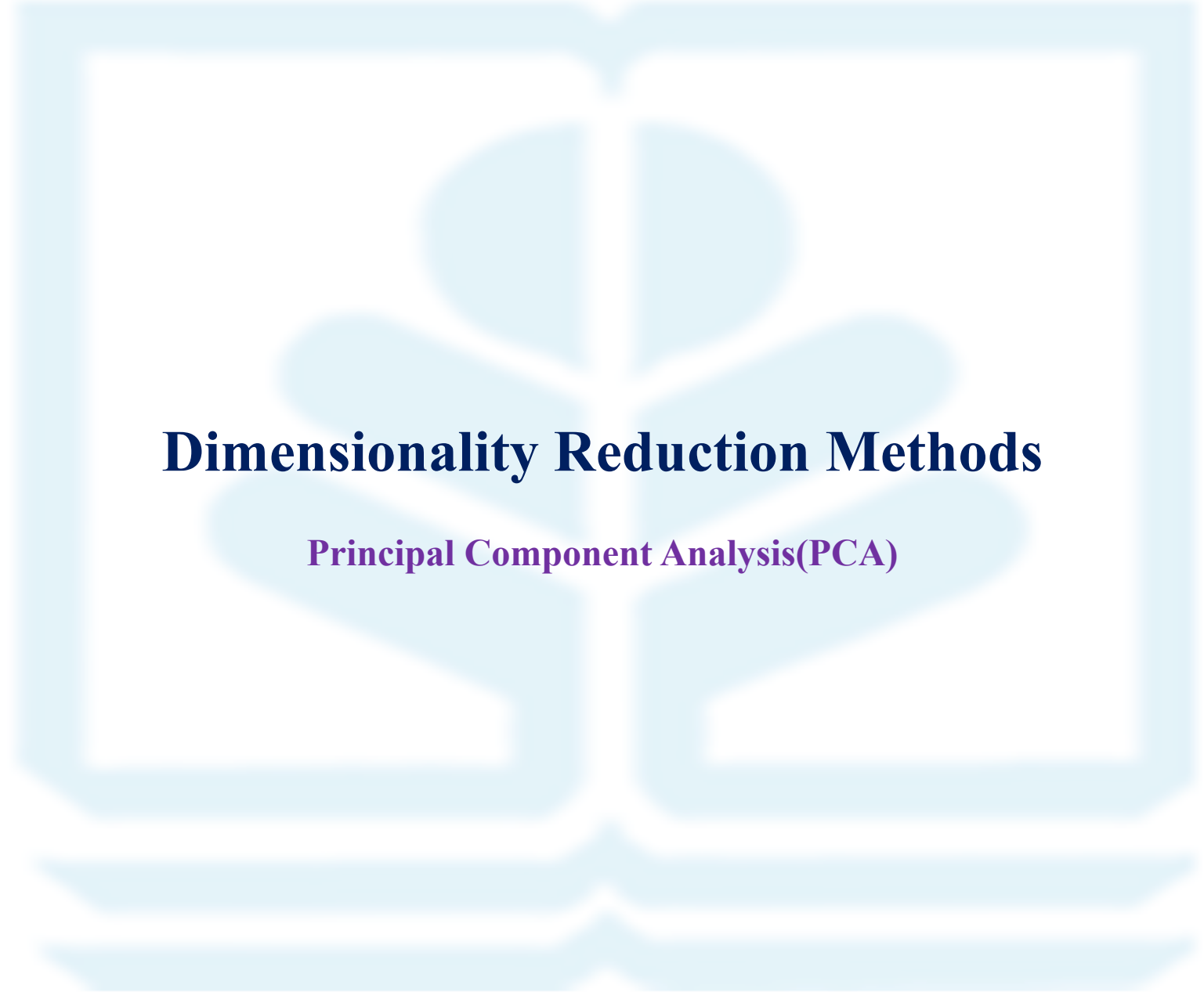
Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

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.



The logo of the University of Mazandaran is a large, light blue watermark in the background. It features a stylized tree with five main branches, each ending in a circular leaf. The tree is enclosed within a square frame with rounded corners. Below the square frame are three horizontal lines of varying lengths, suggesting a base or a series of steps.

Dimensionality Reduction Methods

Principal Component Analysis(PCA)

❖ **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.

PCA Algorithm

Principal Component Analysis(PCA)

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

Output:

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

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

		<i>Features</i>			
		$x_1^{(i)}$	$x_2^{(i)}$	...	$x_n^{(i)}$
<i>Samples</i>	$\mathbf{x}^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	...	$x_n^{(1)}$
	$\mathbf{x}^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	...	$x_n^{(2)}$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	$\mathbf{x}^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	...	$x_n^{(m)}$

PCA Algorithm

Principal Component Analysis(PCA)

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

Output:

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

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

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

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.

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`

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

`from sklearn.preprocessing import MinMaxScaler`

Covariance

❖ Variance and Covariance:

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

➤ Variance:

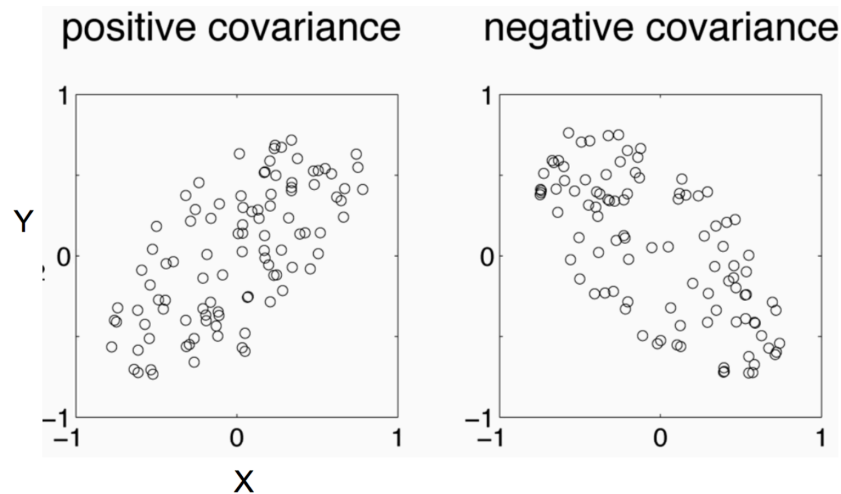
Measures the deviation from the mean for points in one dimension.

➤ 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

PCA Algorithm

Principal Component Analysis(PCA)

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

Output:

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

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

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

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

Step 2. Compute eigenvectors and eigenvalue of Σ :

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

Step 3. Select top k eigenvectors.

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

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

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

PCA Algorithm

Example 1.

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

Dataset matrix

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

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

PCA Algorithm

Step 0. (Pre-processing)

Example 1.

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

PCA Algorithm

Step 0. (Pre-processing)

Example 1.

Cont.

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

Dataset matrix

$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
-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

$$\begin{aligned}\mu_1 &= 4 \\ \sigma_1 &= 3 \\ x_1^{(5)} &= 8\end{aligned}$$

$$\begin{aligned}x_1^{(5)} &\leftarrow \frac{x_1^{(5)} - \mu_1}{\sigma_1} \\ &= 1.49071\end{aligned}$$

$$\begin{aligned}\mu_4 &= 3.4 \\ \sigma_4 &= 2.30217 \\ x_4^{(1)} &= 4 \\ x_4^{(1)} &\leftarrow \frac{x_4^{(1)} - \mu_4}{\sigma_4} \\ &= 0.29138\end{aligned}$$

PCA Algorithm

Step 0. (Pre-processing)

Example 1.

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

Standardized Dataset

Cont.

Tian-12-PCA.pdf

PCA Algorithm

Step 0. (Pre-processing)

Example 1.

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

Standardized Dataset

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

Cont.

Tian-12-PCA.pdf

PCA Algorithm

Example 1.

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

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

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



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

PCA Algorithm

Example 1.

Step 2. Compute eigenvectors and eigenvalue of Σ

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

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

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

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

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

$$\lambda_1 = 2.5158 \quad e_1 = [-0.1620 \quad 0.5240 \quad 0.5859 \quad 0.5965]^T$$

$$\lambda_2 = 1.0653 \quad e_2 = [-0.9171 \quad 0.2069 \quad -0.3205 \quad -0.1159]^T$$

$$\lambda_3 = 0.3939 \quad e_3 = [0.3071 \quad 0.8173 \quad -0.1883 \quad -0.4497]^T$$

$$\lambda_4 = 0.0250 \quad e_4 = [0.1962 \quad 0.1206 \quad -0.7201 \quad 0.6545]^T$$

The first principal component

The Second principal component

The third principal component

The fourth principal component

PCA Algorithm

Example 1.

Step 3. Select top k eigenvectors, $k=2$

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

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

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

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

for $i=1,2,\dots, m$

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

PCA Algorithm

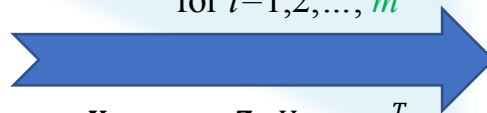
Example 1.

❖ *Reconstruction from compressed representation, $k=2$*

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

$$\mathbf{X}_{approx}^{(i)} = \mathbf{U}_{reduced} \times \mathbf{Z}^{(i)},$$

for $i=1, 2, \dots, m$



$$\mathbf{X}_{approx} = \mathbf{Z} \times \mathbf{U}_{reduced}^T$$

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

PCA Algorithm

Example 2.

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

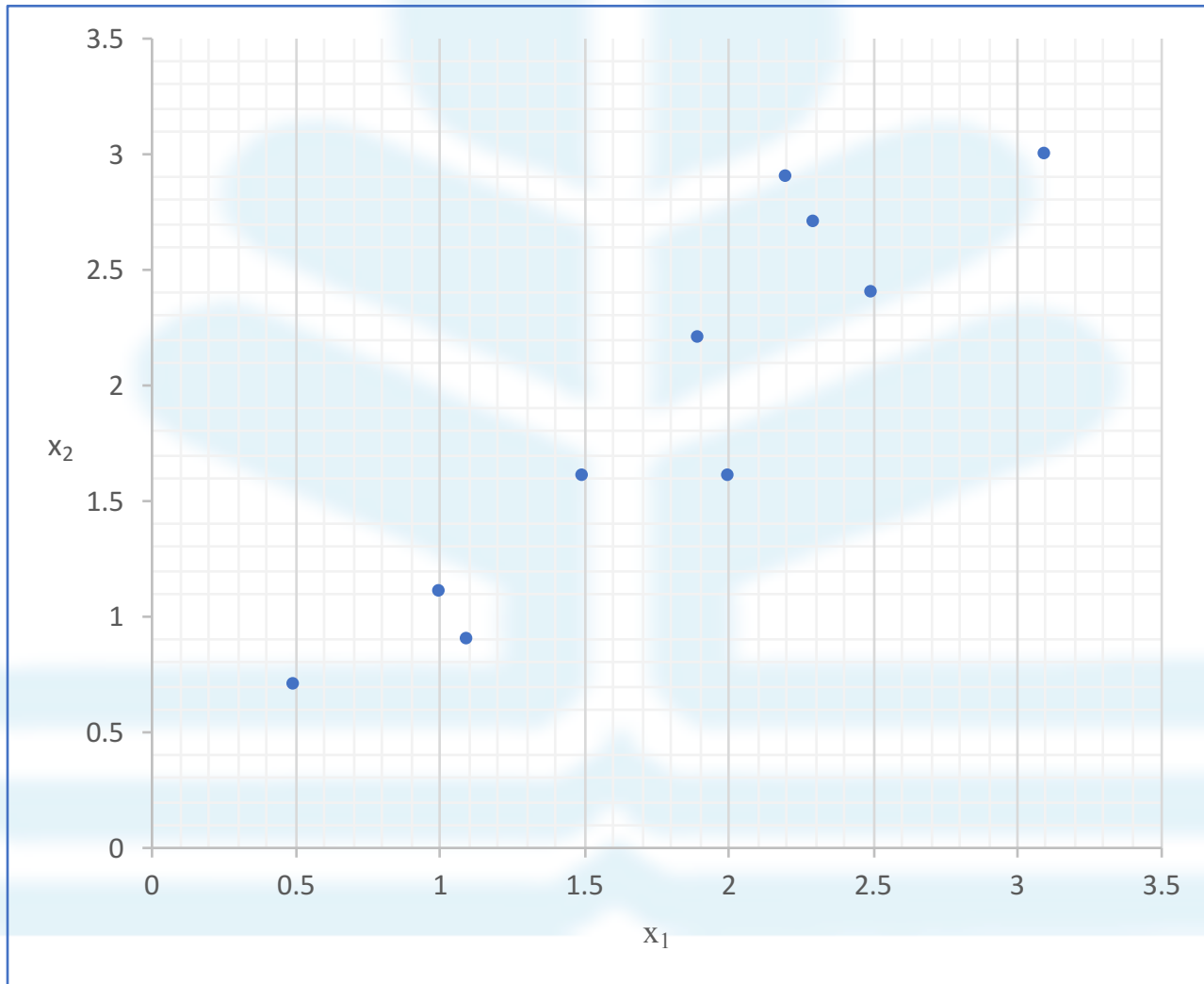
Original Dataset

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

PCA Algorithm

Example 2.

Dataset



Cont.

PCA Algorithm

Example 2.

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

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

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

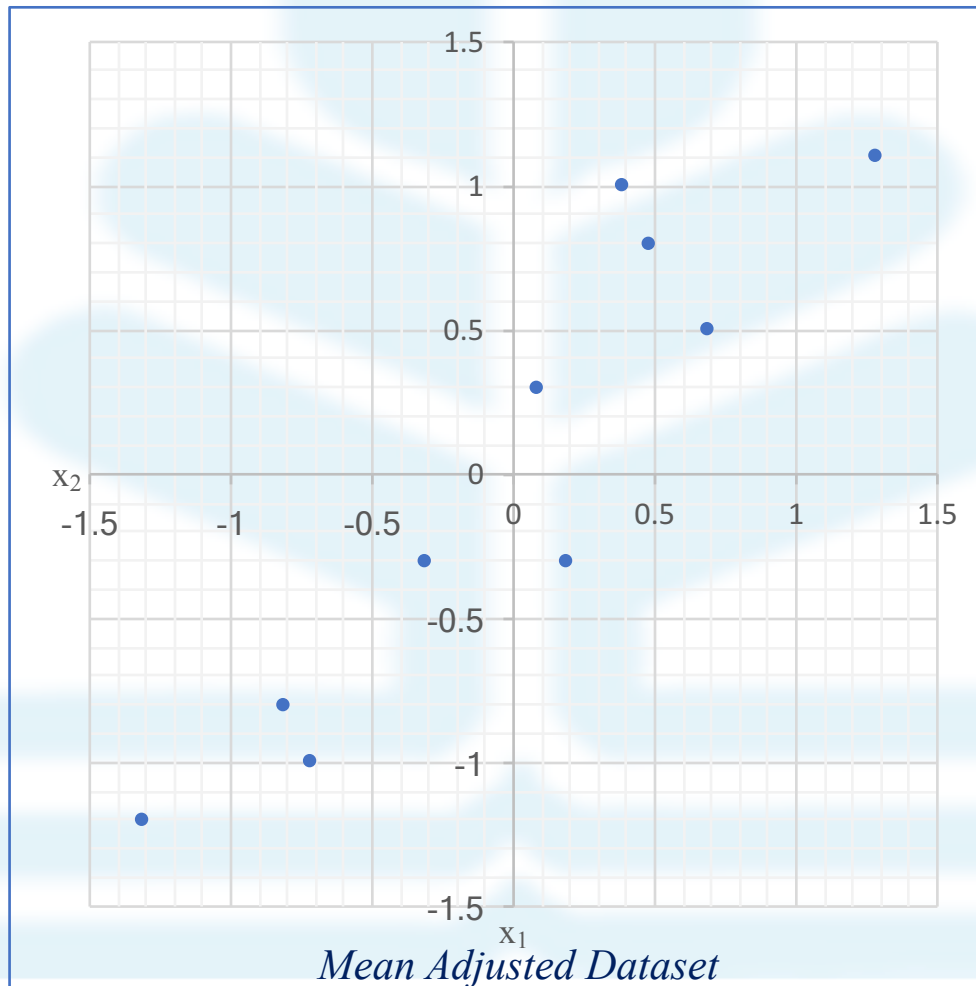
Mean Adjusted Dataset

PCA Algorithm

Cont.

Example 2.

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



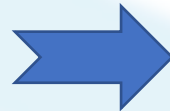
PCA Algorithm

Example 2.

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

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

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



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

PCA Algorithm

Example 2.

Step 2. Compute eigenvectors and eigenvalue of Σ

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



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

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

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

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



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

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

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

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

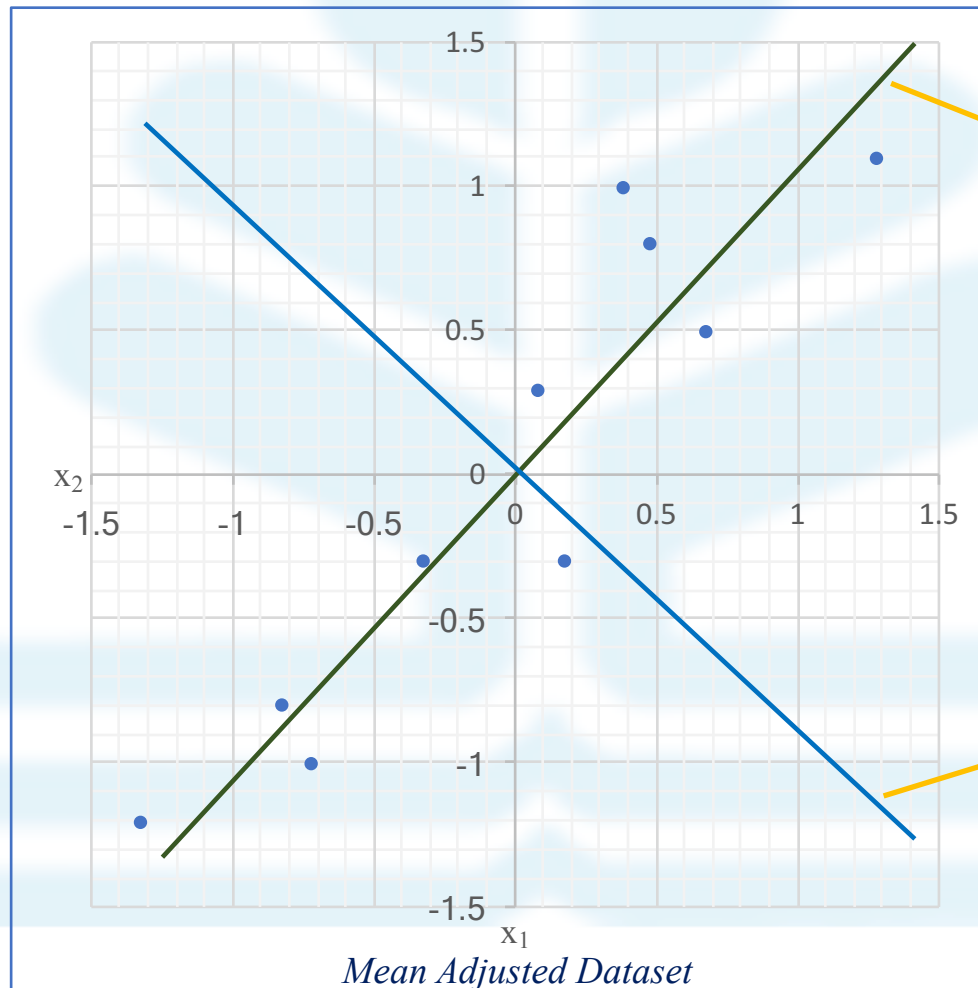
PCA Algorithm

Cont.

Example 2.

Step 2. Compute eigenvectors and eigenvalue of Σ

Mean Adjusted Dataset and eigenvectors of Σ



The first principle component

The Second principle component

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:

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

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

for $i=1,2,\dots, m$

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

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:

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

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

for $i=1,2,\dots, m$

	$\mathbf{z}_1^{(i)}$
$\mathbf{z}^{(1)}$	-0.8280
$\mathbf{z}^{(2)}$	1.7776
$\mathbf{z}^{(3)}$	-0.9922
$\mathbf{z}^{(4)}$	-0.2742
$\mathbf{z}^{(5)}$	-1.6758
$\mathbf{z}^{(6)}$	-0.9129
$\mathbf{z}^{(7)}$	0.0991
$\mathbf{z}^{(8)}$	0.0991
$\mathbf{z}^{(9)}$	0.4380
$\mathbf{z}^{(10)}$	1.2238

PCA Algorithm

Example 2.

❖ *Reconstruction from compressed representation, $k=1$*

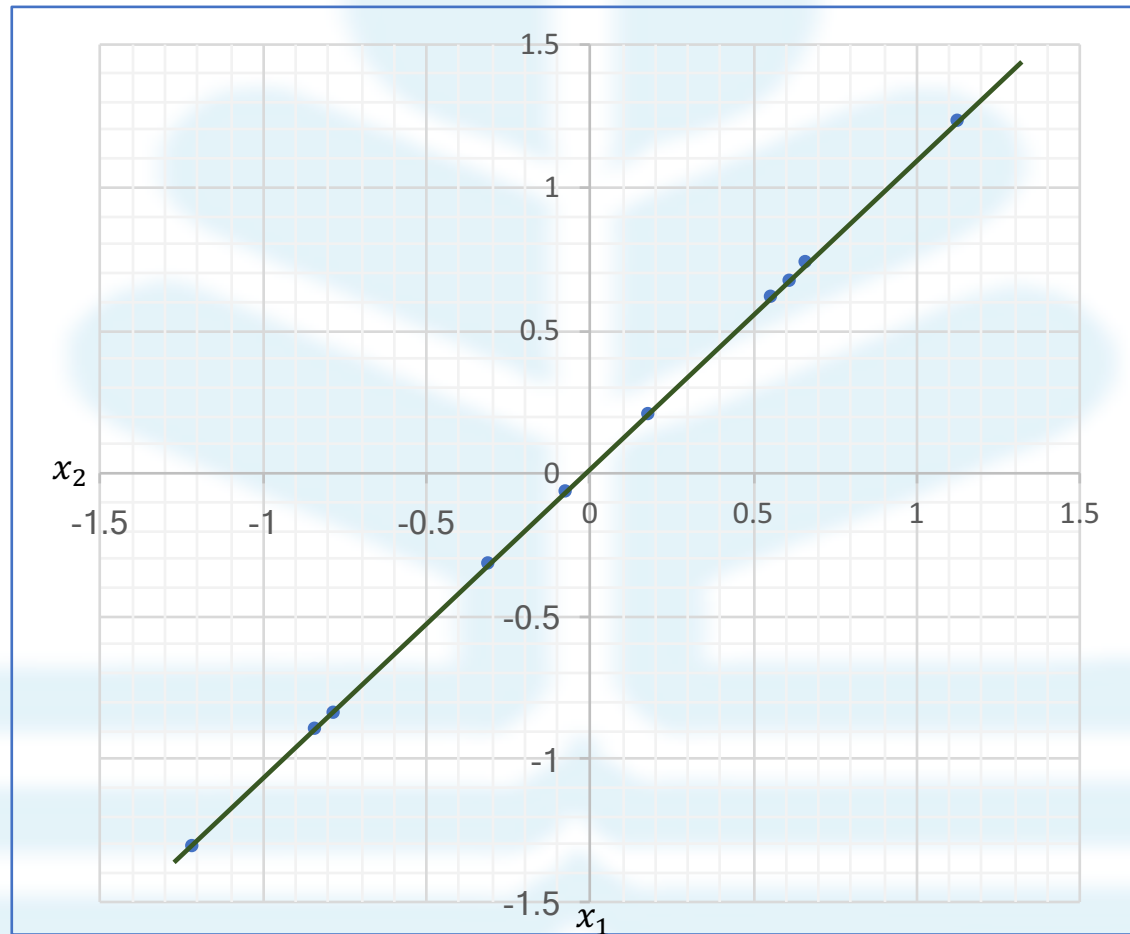
$$\mathbf{z}^{(1)} = \begin{bmatrix} -0.8280 \\ 1.7776 \\ -0.9922 \\ -0.2742 \\ -1.6758 \\ -0.9129 \\ 0.0991 \\ 1.1446 \\ 0.4380 \\ 1.2238 \end{bmatrix} \xrightarrow{\mathbf{x}_{approx}^{(i)} = U_{reduced} \times \mathbf{z}^{(i)}, \text{ for } i=1,2,\dots, m} \mathbf{x}_{approx}^{(1)} = \begin{bmatrix} 0.5613 & 0.6087 \\ -1.2050 & -1.3068 \\ 0.6726 & 0.7294 \\ 0.1859 & 0.2016 \\ 1.1360 & 1.2320 \\ 0.6189 & 0.6712 \\ -0.0672 & -0.0729 \\ -0.7759 & -0.8415 \\ -0.2969 & -0.3220 \\ -0.8296 & -0.8997 \end{bmatrix}$$

Reconstruction from compressed representation

Cont.

Example 2.

❖ *Reconstruction from compressed representation, $k=1$*



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]])
```

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

```
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)
```

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]]
```

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

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

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

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
```

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

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
```

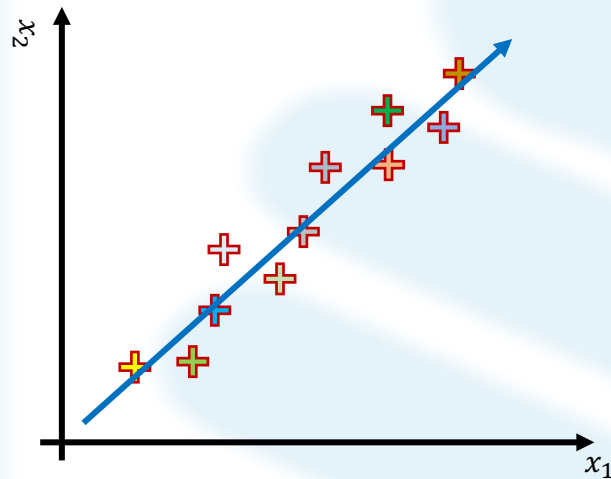
```
                             , columns = ['principal component ' + str(i+1) for i in range(n_components)])
```

```
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

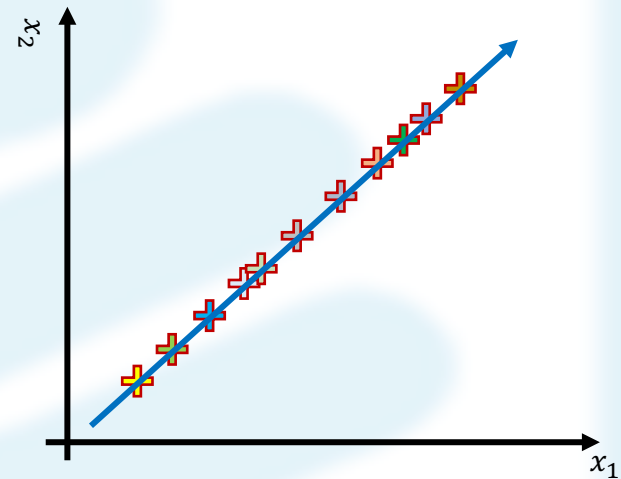
Reconstruction from compressed representation

Reduce data from 2D to 1D



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

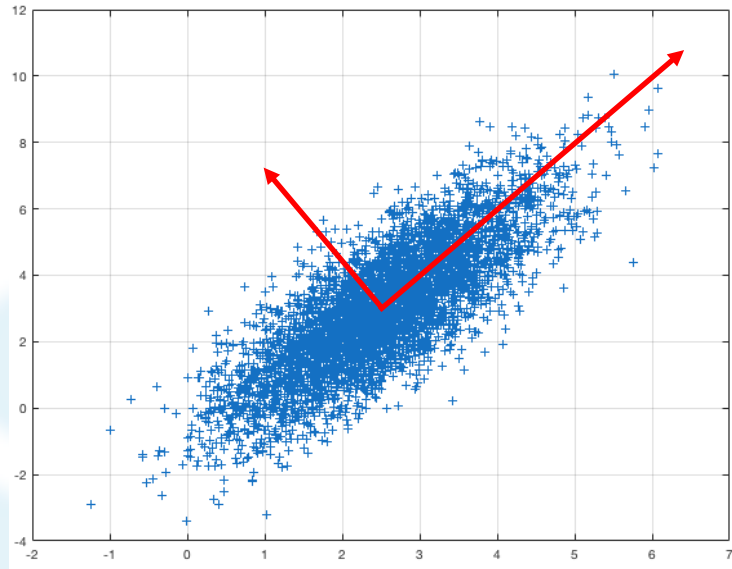
for $i=1,2,\dots, m$



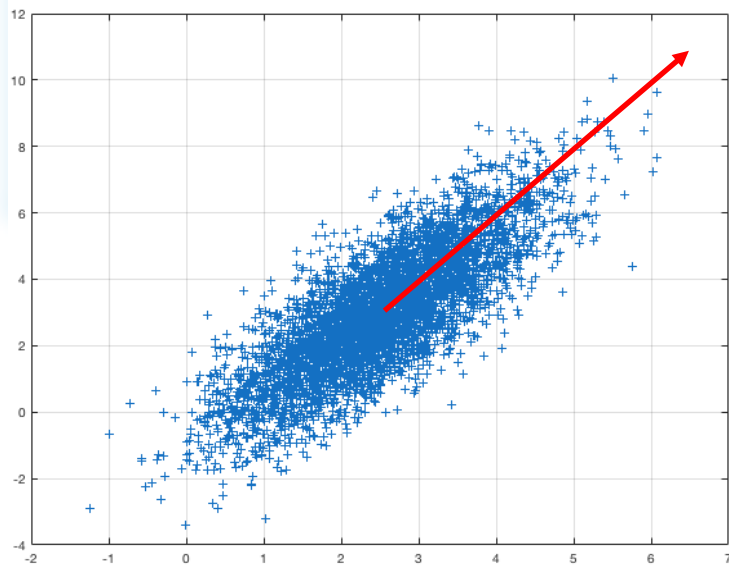
$$\mathbf{x}_{approx}^{(i)} = U_{reduced} \times \mathbf{z}^{(i)},$$

for $i=1,2,\dots, m$

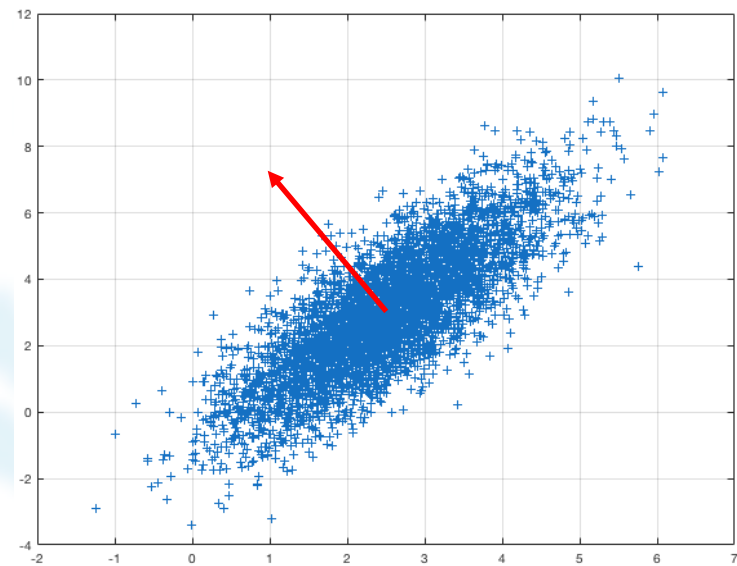
PCA



1st PCA axis



2nd PCA axis



Choosing k (number of principal components)

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

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

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

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

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

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

If $R(\mathbf{x}, \mathbf{x}_{approx}, k) \leq 0.05$, then 95% of variance is retained.

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

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

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.

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

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

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

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

Kernel PCA

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

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

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

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

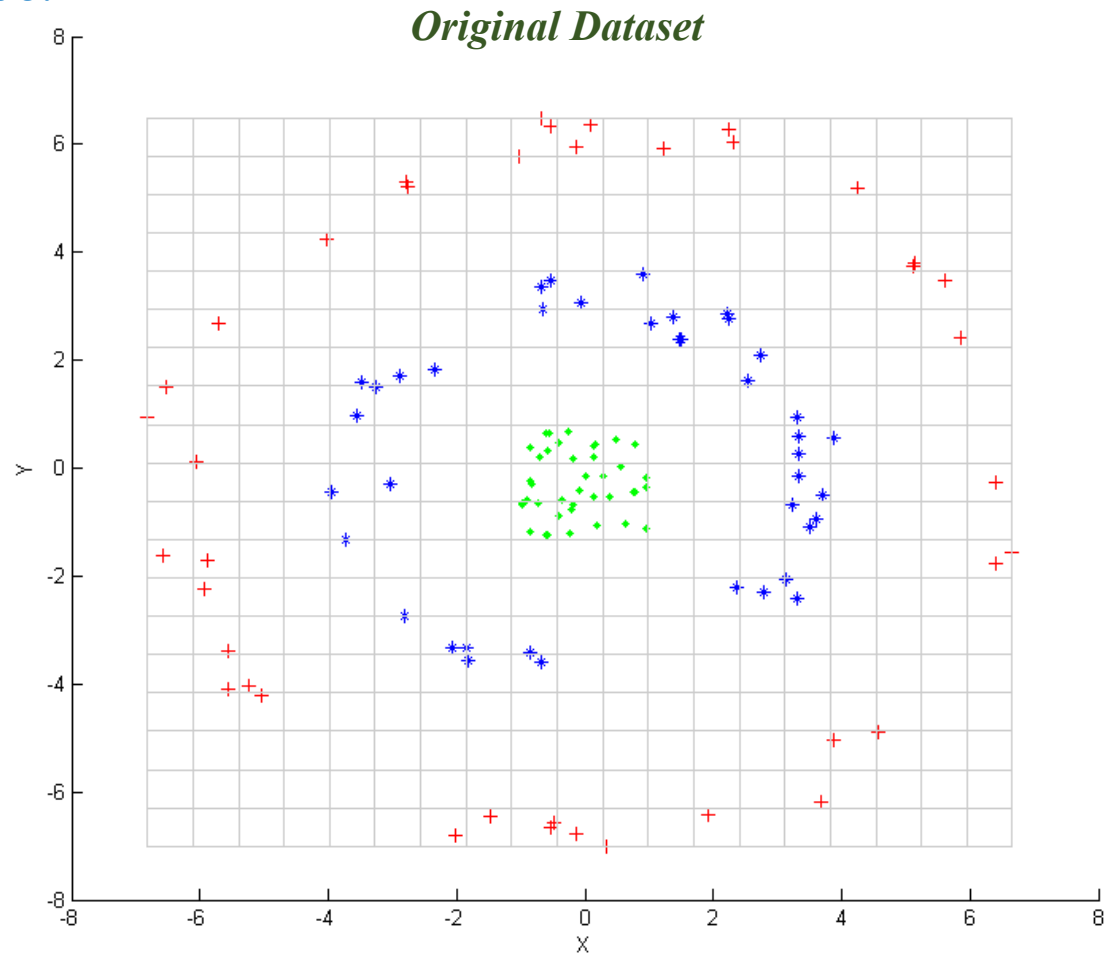
The data should be centered in the feature space, too! But this is manageable.

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

```
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)
```

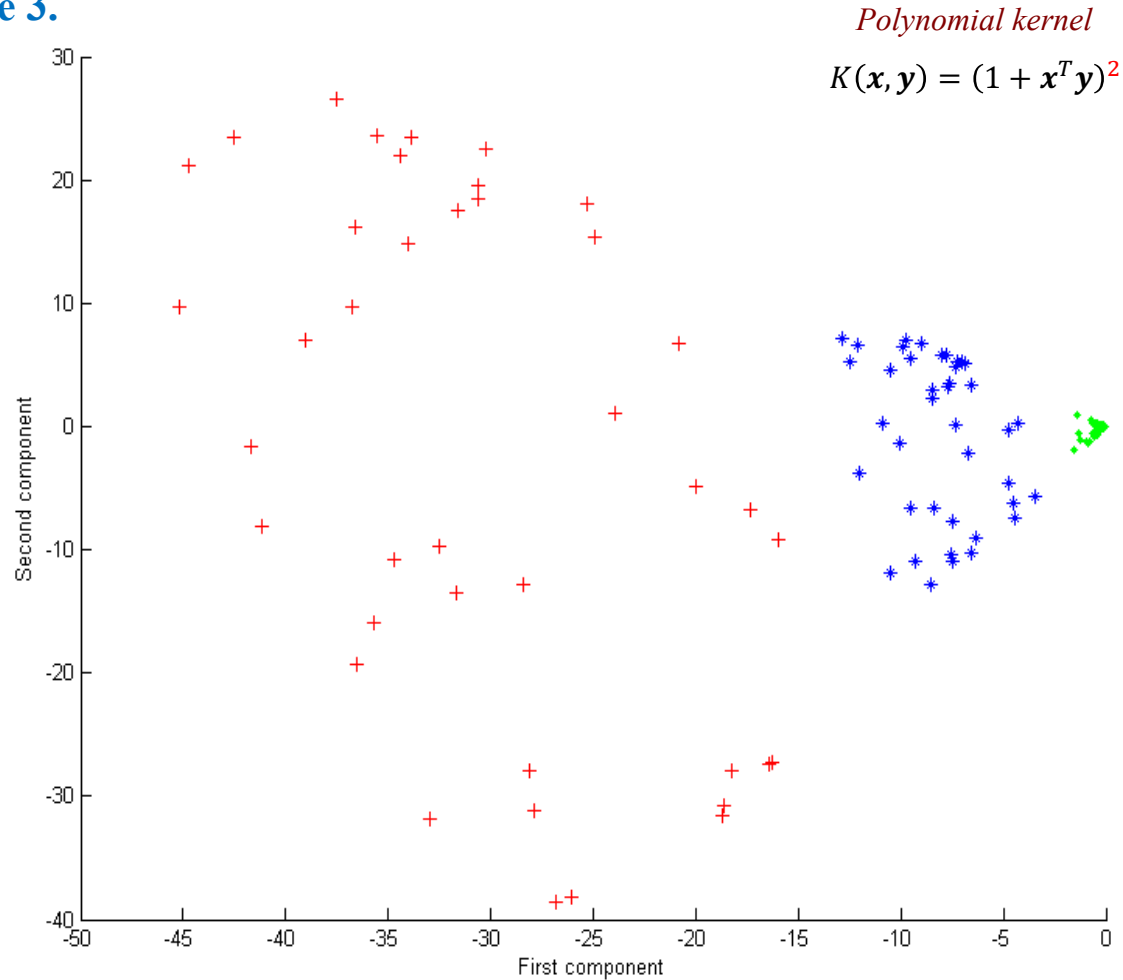

Input points before kernel PCA

Example 3.



Output after kernel PCA

Example 3.



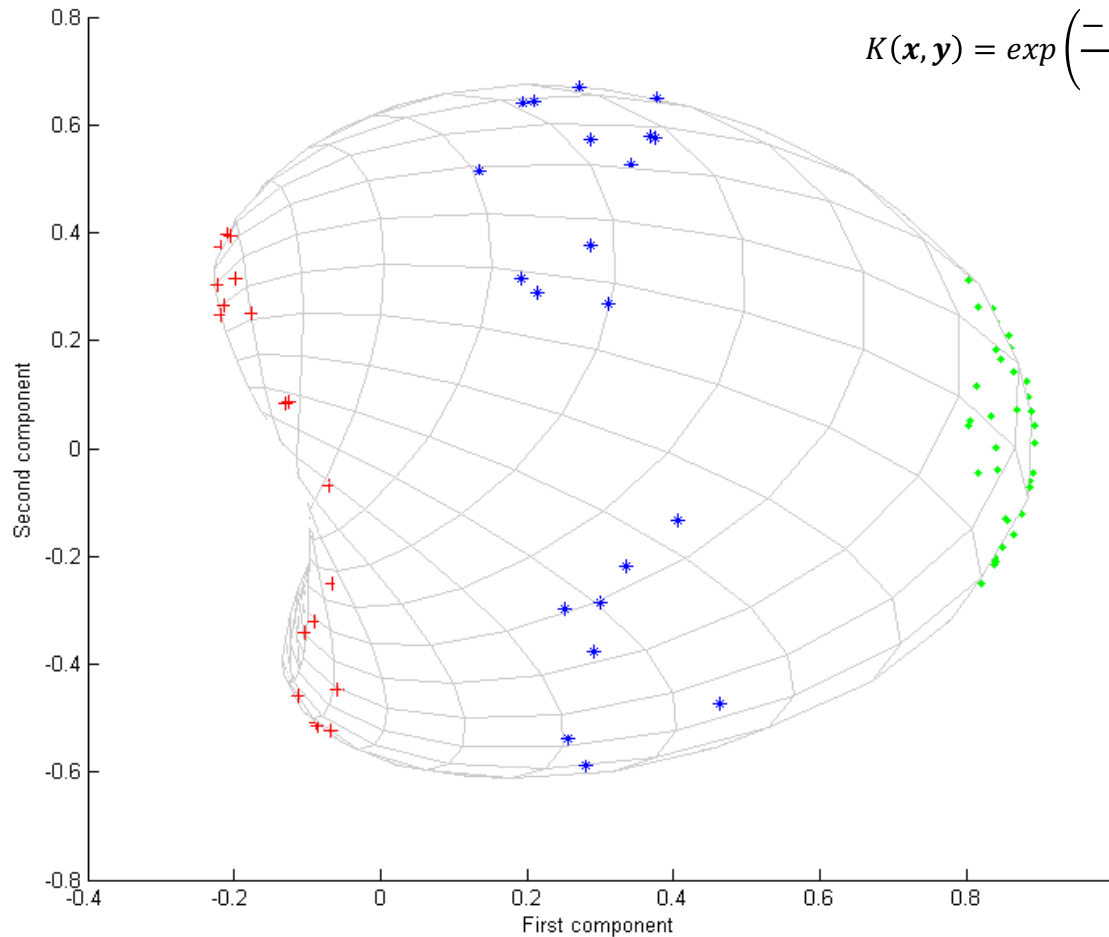
The three groups are distinguishable using the first component only

Kernel PCA

Example 3.

Gaussian kernel

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



The three groups are distinguishable using the first component only

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