Machine Learning

Amin Golzari Oskouei

a.golzari@azaruniv.ac.ir

a.golzari@tabrizu.ac.ir

https://github.com/Amin-Golzari-Oskouei

Azarbaijan Shahid Madani University
2023

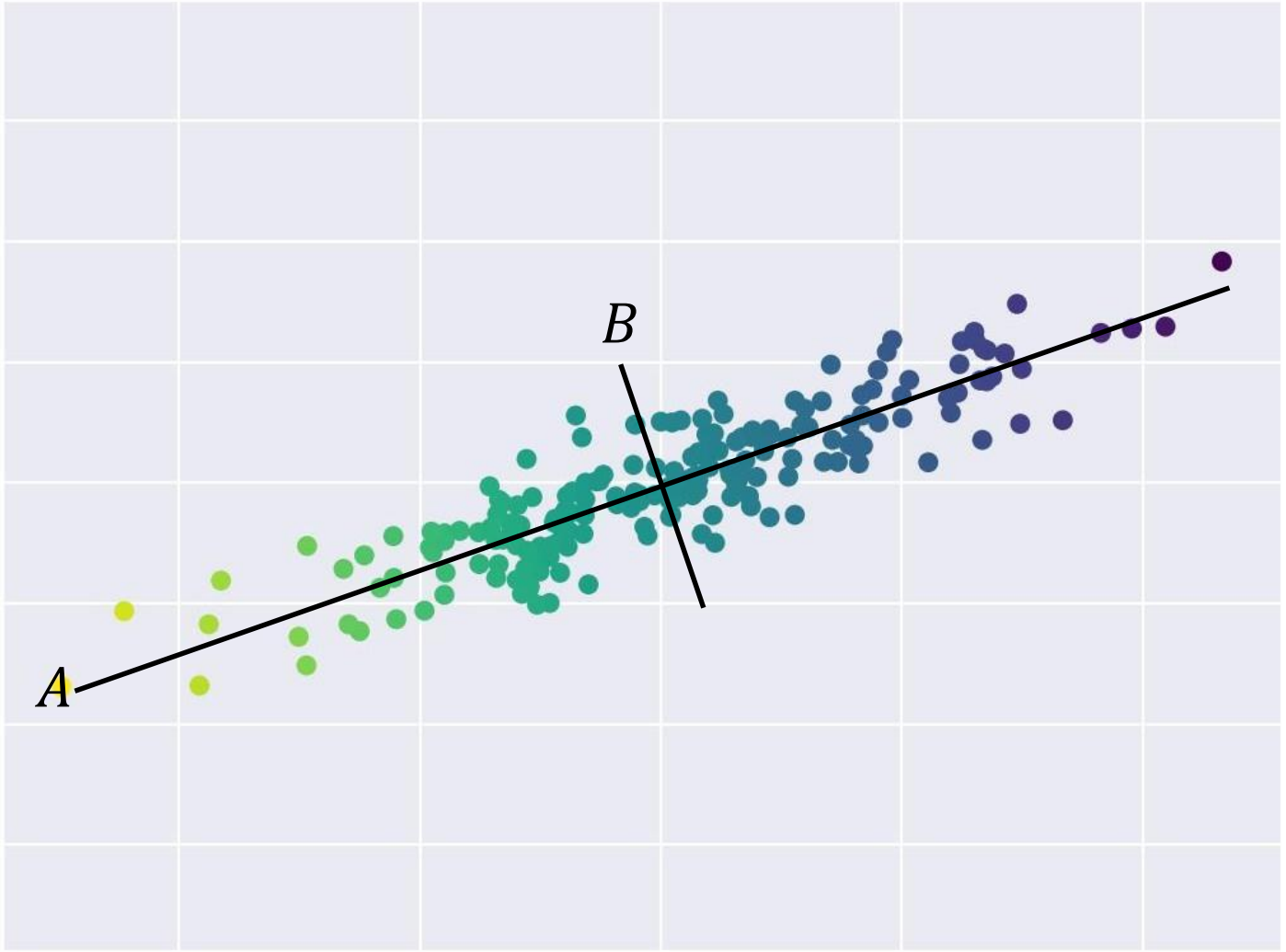
Dimensionality reduction

Dimensionality reduction

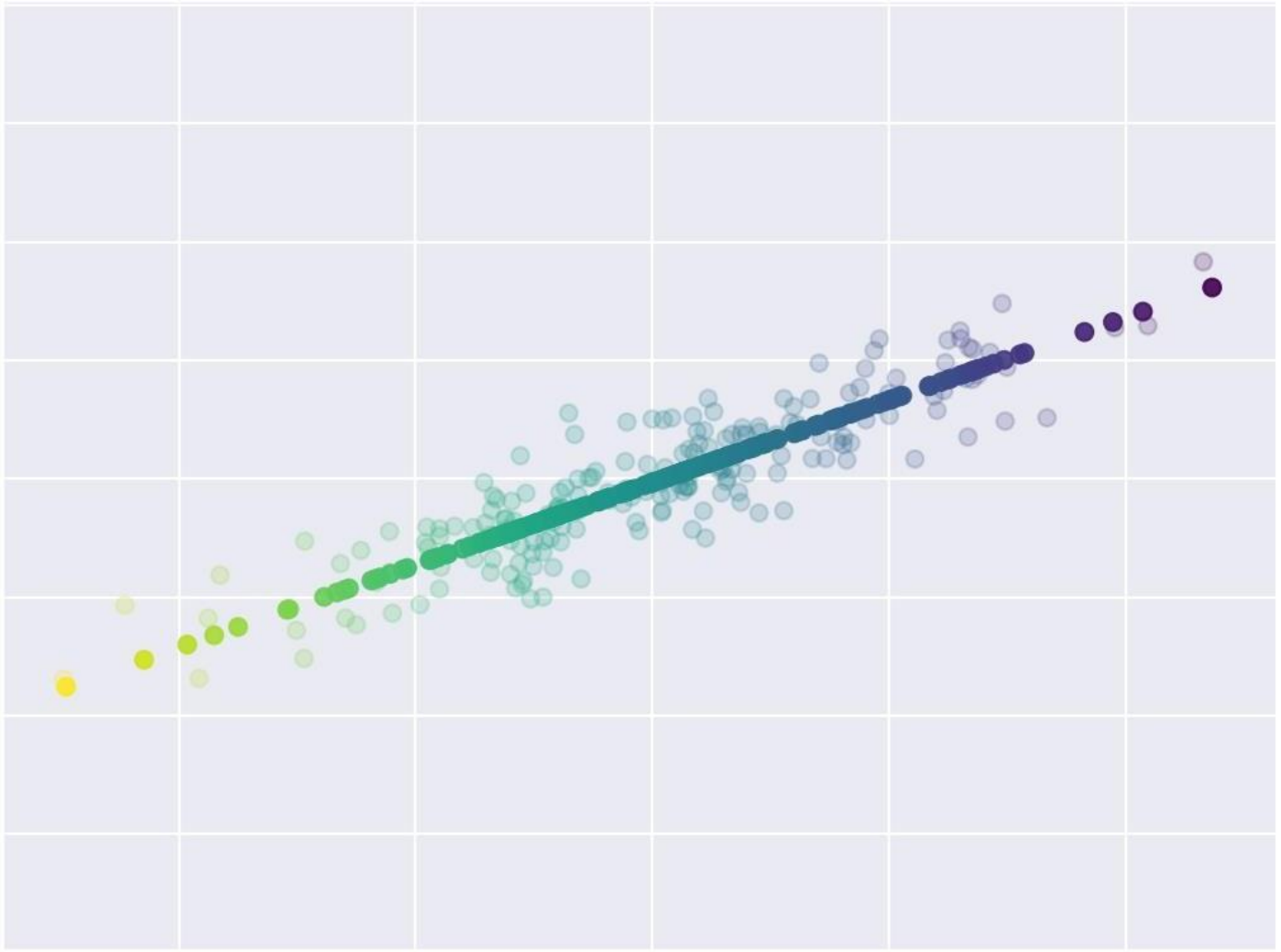
- Motivation
  - displaying data
  - data compression
    - reduction of consuming memory
    - using easier of data set
    - reduction of loss function of a variety of algorithm
    - noise elimination and increasing the accuracy of a learning algorithm
    - to simplify the results preception
- Methods
  - PCA(principal component analysis)
  - Factor analysis
  - independent component analysis

PCA(principal component analysis)

PCA(principal component analysis)



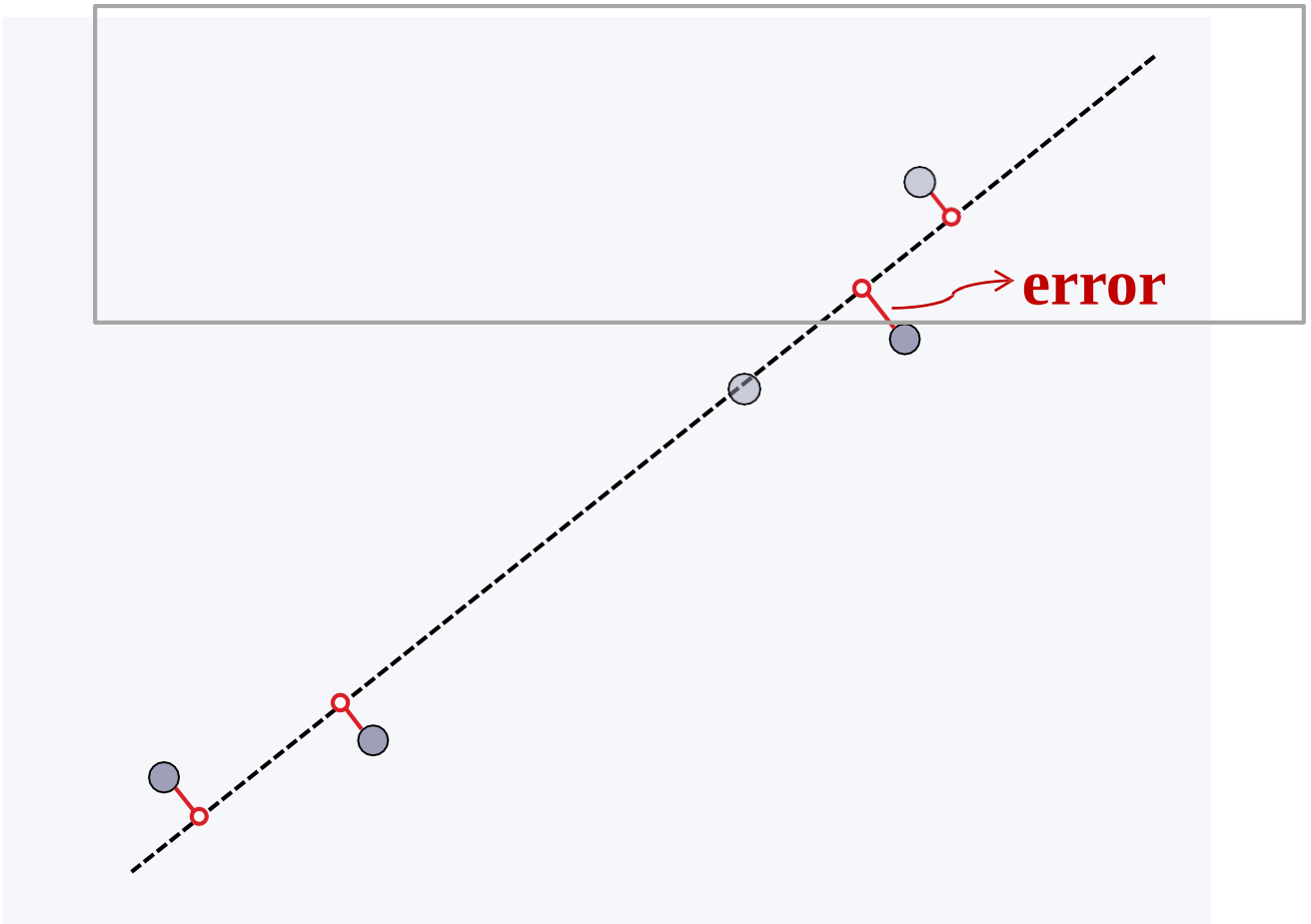
PCA(principal component analysis)



PCA: defining problems

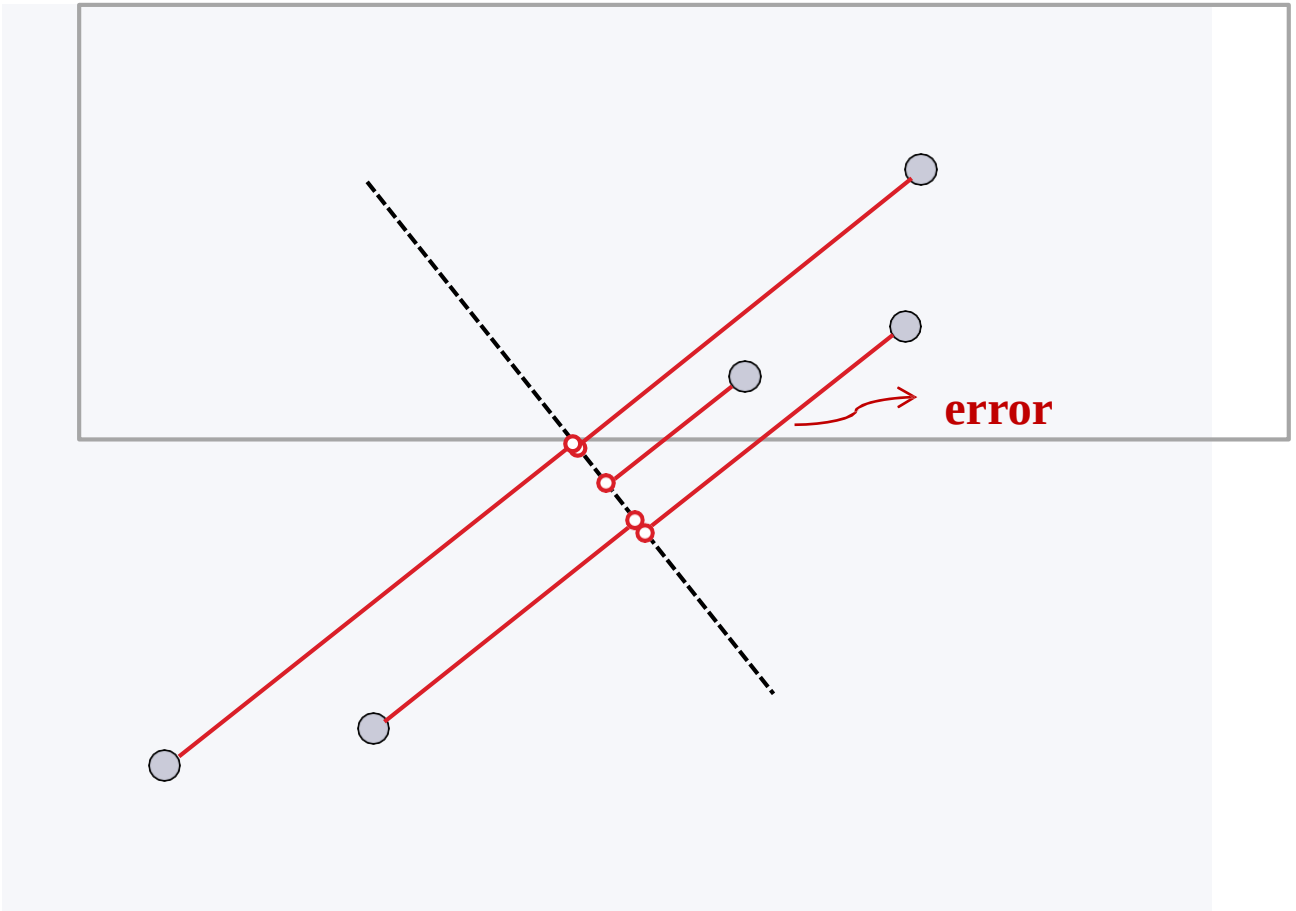
Defining problems

- **Purpose.** Minimization of the sum of squares of radiation error



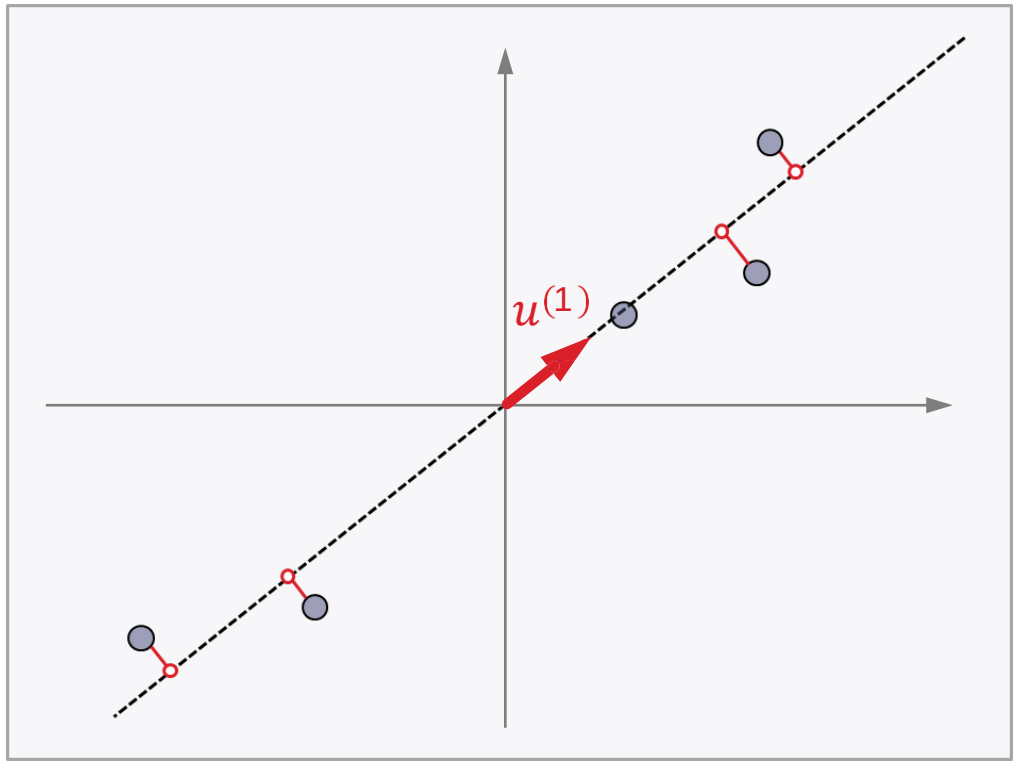
Defining problems

- **Purpose.** Minimization of the sum of squares of radiation error



Defining problems

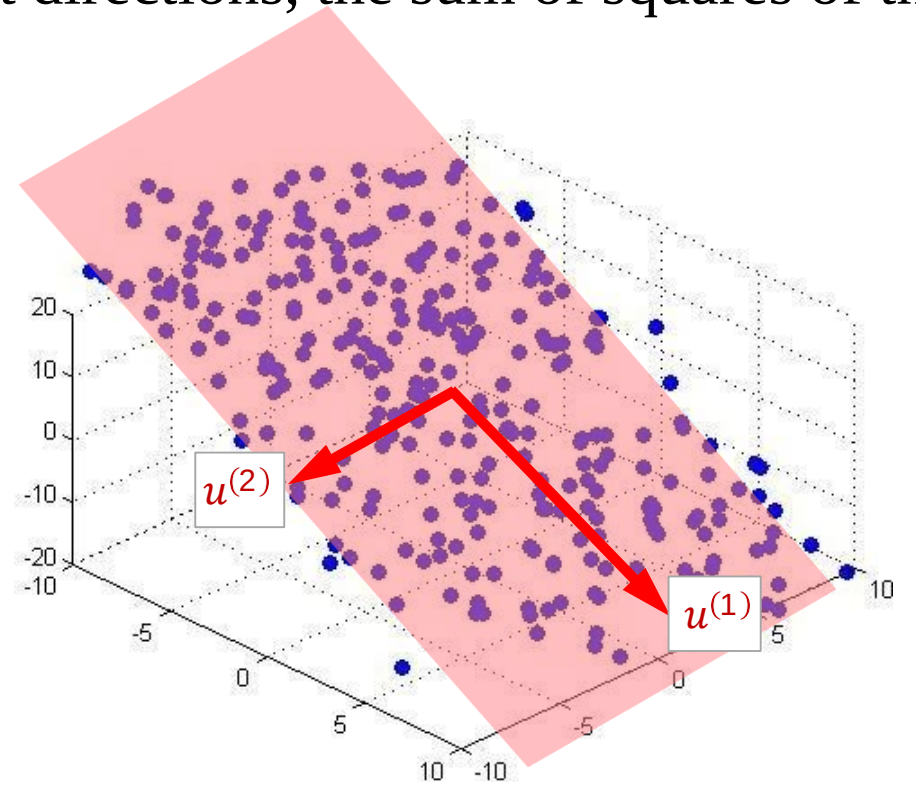
- Dimensionality reduction from 2 to 1.Finding one direction like
  by depicting the points in that direction, the sum of squares of the error is minimized.



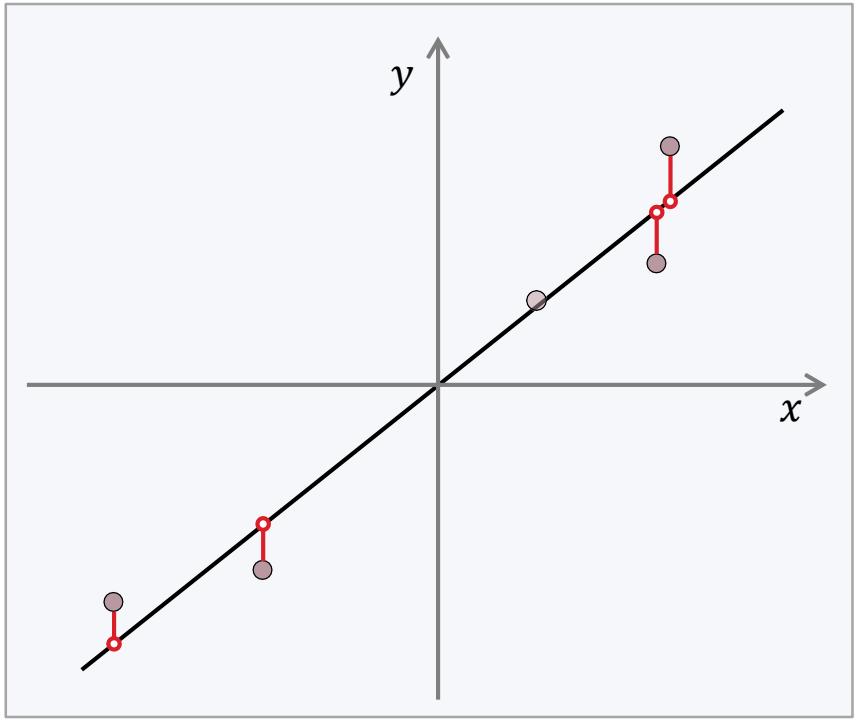
Defining problems

- Dimensionality reduction from n to k. Finding K Orthogonal vector

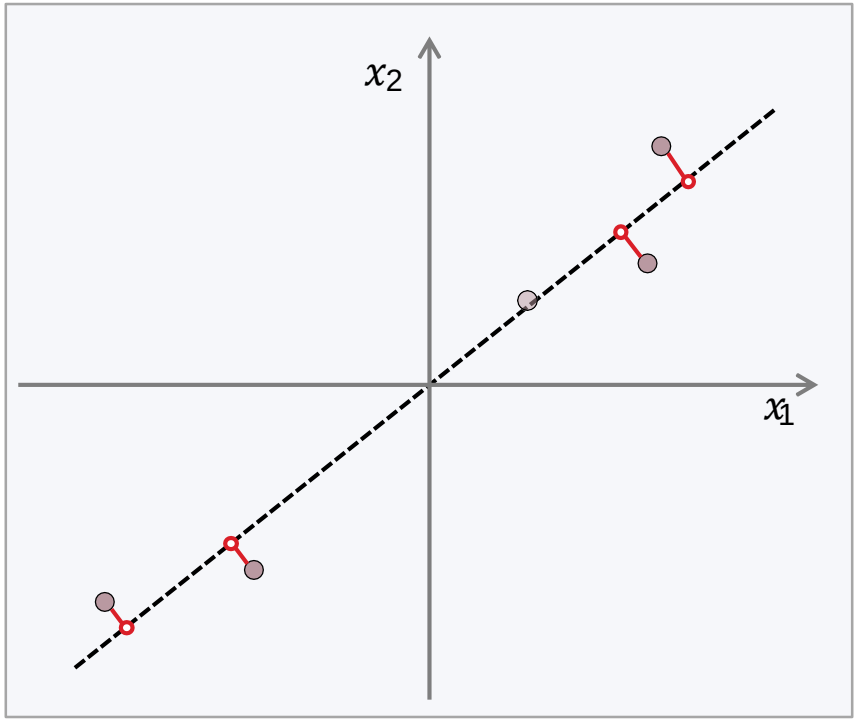
by depicting the points in that directions, the sum of squares of the error is minimized.



PCA is regression ?



Linear regression



Principal component analysis

PCA Algorithm

PCA Algorithm: Preprocessing

- training set

$$\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\} \quad , \quad x^{(i)} \in \mathbb{R}^n$$

- preprocessing

(1) Remove the average

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

(2) Scaling

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

s_j ← standard deviation in j

PCA algorithm: dimension reduction

- Reduce data dimensions
  - calculating < covariance matrix >

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

  - calculating < eigenvectors> covariance matrix

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

  - choose k vectors from U matrix

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

PCA algorithm

- Calculating new data with K dimensions

$$z^{(i)}_{k\times 1} = \begin{bmatrix} | & | & \cdots & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & \cdots & | \end{bmatrix}^T \times x^{(i)}_{n\times 1}$$

$$= \begin{bmatrix} - & u^{(1)} & - \\ - & u^{(2)} & - \\ \vdots & \vdots & \vdots \\ - & u^{(k)} & - \end{bmatrix}_{k\times n} \times x^{i(\ )}_{n\times 1}$$

Implementation in Python

PCA algorithm

After removing the average and Scaling if needed

```
def PCA(X, k):

    m = X.shape[0]

    Sigma = (X.T @ X) / m

    U, S, V = svd(Sigma)

    U_reduced = U[:, :k]

    Z = X @ U_reduced

    return Z
```

Calculating covariance matrix

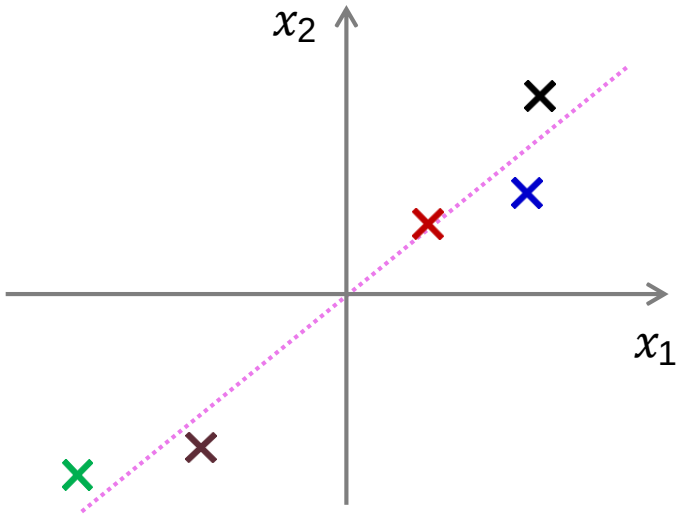
Calculating the decomposition of single values

Choosing the k of The first component

Calculating new data with K dimension

Example: dimensions reduction

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

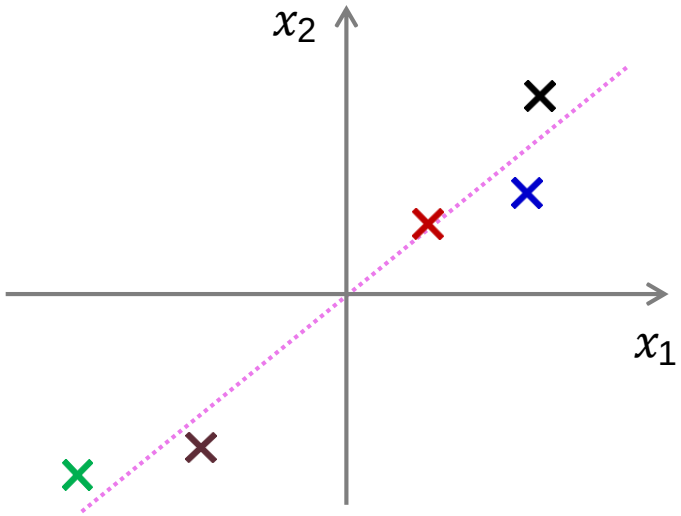


primary data (2D)

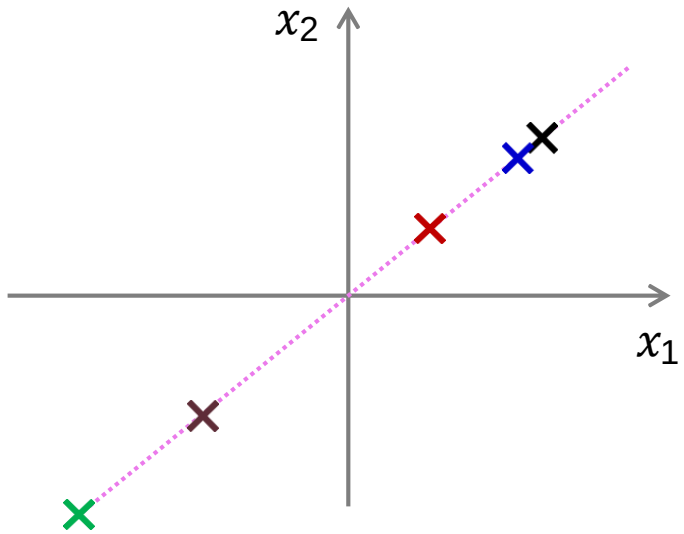
New data (1D)

Example: dimensions reduction

$$X_{recovered} = Z * U^T{}_{reduced} + means$$



primary data (2D)



Reconstructed data

PCA algorithm: remove the average


```
X = np.array([[1, 1, 1, 0, 0],
              [2, 2, 2, 0, 0],
              [1, 1, 1, 0, 0],
              [5, 5, 5, 0, 0],
              [1, 1, 0, 2, 2],
              [0, 0, 0, 3, 3],
              [0, 0, 0, 1, 1]])


mu = X.mean(axis=0)
X_norm = X - mu
```

primary data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

PCA algorithm: Calculation of eigenvectors


```
m = X.shape[0]
Sigma = (X_norm.T @ X_norm) / m  U, S,
V = np.linalg.svd(Sigma)


print(S)
```

[8.72e+00 1.58e+00 6.69e-02 4.79e-16 1.35e-47]

$$S = \begin{bmatrix} s_{11} & 0 & \cdots & 0 \\ 0 & s_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_{nn} \end{bmatrix}$$

PCA algorithm: dimension reduction


```
U_red = U[:, :3]


X_proj = X_norm * U_red
```

```
[[ 0.17    1.37   -0.01]
 [-1.44    0.74   -0.02]
 [ 0.17    1.37   -0.01]
 [-6.28   -1.17   -0.06]
 [ 1.76   -1.1     0.57]
 [ 3.33   -1.92   -0.35]
 [ 2.3     0.7    -0.11]]
```

PCA algorithm: Reconstruction of primary data


```
X_approx = X_proj @ U_red + mu
```

```
[[  1.00   1.00   1.00   0.00   0.00]
    2.00   2.00   2.00   0.00   0.00]
    1.00   1.00   1.00   0.00   0.00]
    5.00   5.00   5.00  -0.00  -0.00]
    1.00   1.00   0.00   2.00   2.00]
    0.00  -0.00   0.00   3.00   3.00]
   -0.00  -0.00  -0.00   1.00   1.00]]
```

Primary data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

Selecting the number of main components

Selecting the number of main components

- Mean sum of squares of radiation error

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

Keeping 99% of the variance

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\le 0.05$$

Keeping 95% of the variance

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\le 0.10$$

Keeping 90% of the variance

Selecting the number of main components

$k = 0$

repeat

{

$k = k + 1$

try $PCA(X)$ with k components

compute $U_{reduced}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, x_{approx}^{(2)}, \dots, x_{approx}^{(m)}$

} **until** $\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2} \leq 0.01$

An inefficient algorithm

Selecting the number of main components

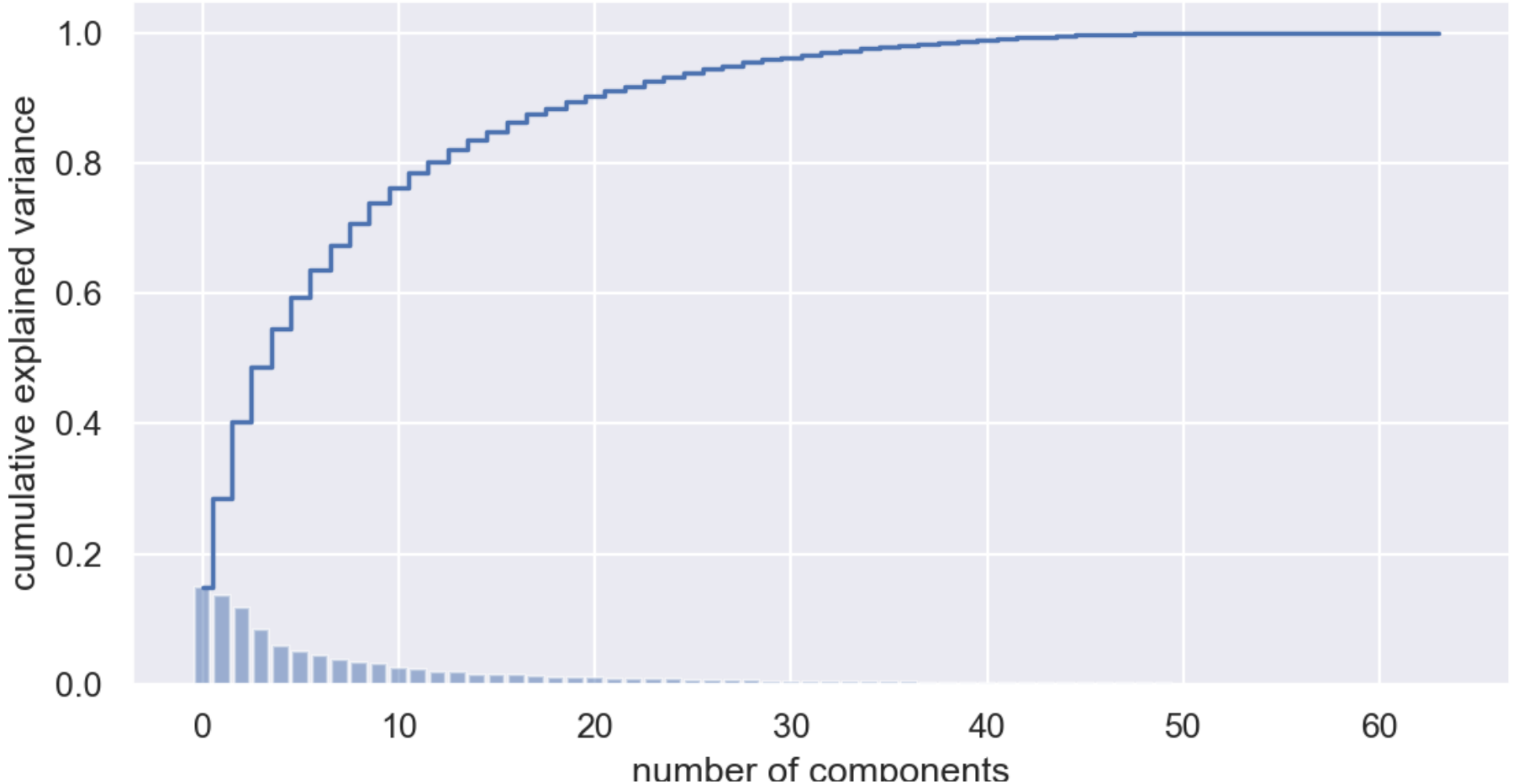

```
m, n = X.shape
X = X - X.mean(axis=0)  Sigma
= (X.T @ X) / m
U, S, V = np.linalg.svd(Sigma)


for k in range(1, n + 1):
        total_var = np.sum(S[:k]) / np.sum(S)  if total_var
        >= 0.99: break
return k
```

$$S = \begin{bmatrix} s_{11} & 0 & \cdots & 0 \\ 0 & s_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_{nn} \end{bmatrix}$$

[8.72e+00 1.58e+00 6.69e-02 4.79e-16 1.35e-47]

Selecting the number of main components: 8*8



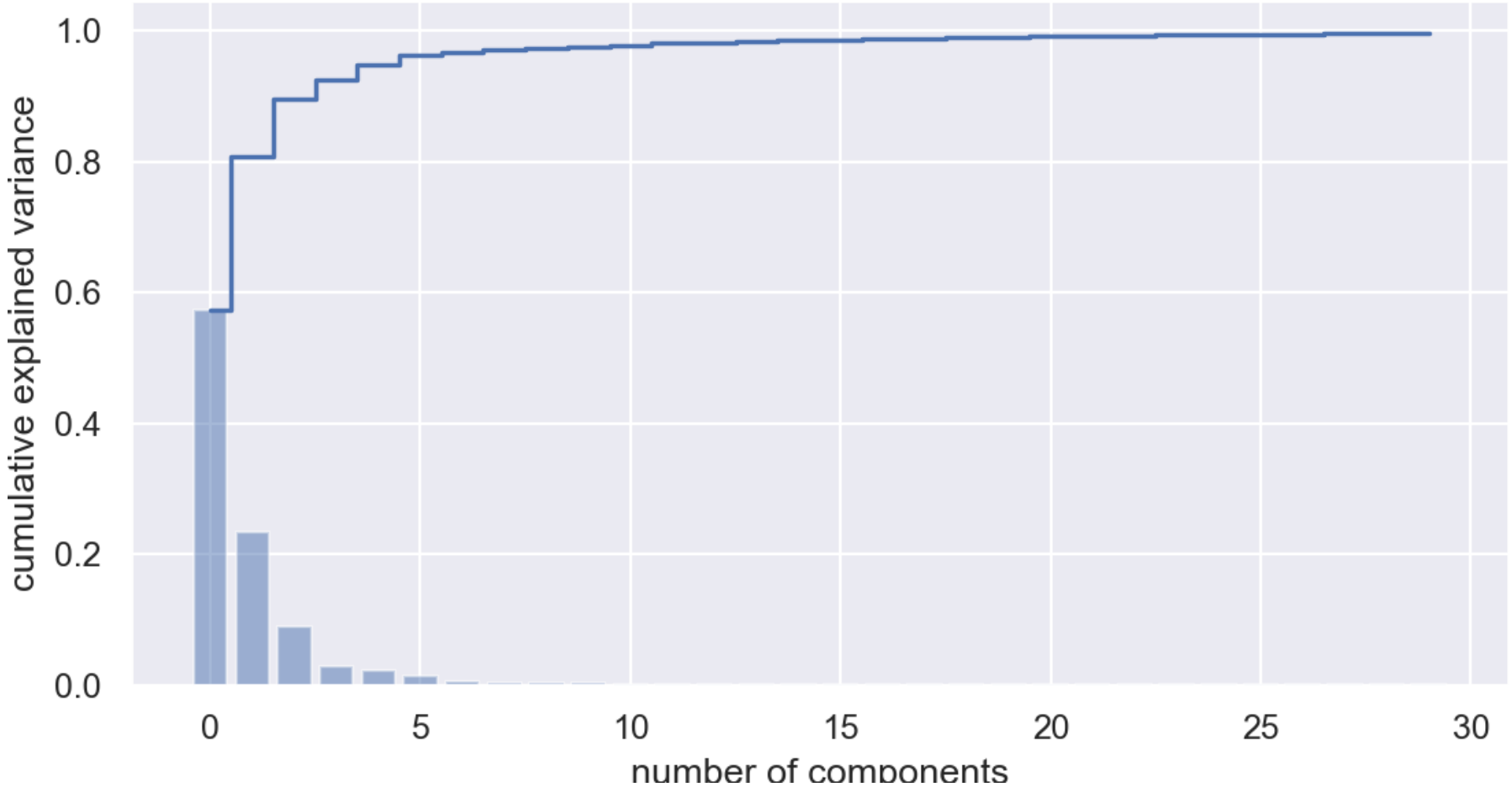
Selecting the number of main components

- Semiconductor data [590 features]

cumulative percentage	Percentage of variance	components
59/2	59/2	1
83/4	24/1	2
92/5	9/2	3
94/8	2/3	4
96/3	1/5	5
96/8	0/5	6
97/1	0/3	7
99/3	0/08	20

http://archive.ics.uci.edu/ml/datasets/SECOM

Selecting the number of main components



Misuse of PCA: Dealing with overfitting

- Incorrect method.
  - Using zi instead of xi reduces the number of features from n to k;
  - As a result, with fewer features, the probability of overfitting
- The right way.
  - PCA does not use information about the output during dimensionality reduction.
  - Use adjustment to deal with overfitting.

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Lastly

- Designing a machine system.
  - Creating training set as $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
  - Using PCA to reduce $x^{(i)}$ and get $z^{(i)}$
  - Running the training phase on $\{(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \dots, (z^{(m)}, y^{(m)})\}$
  - Hypothesis testing using the test set: Mapping $x_{test}^{(i)}$ to $z_{test}^{(i)}$ and calculating $h_\theta(z)$ for each of the test set data.

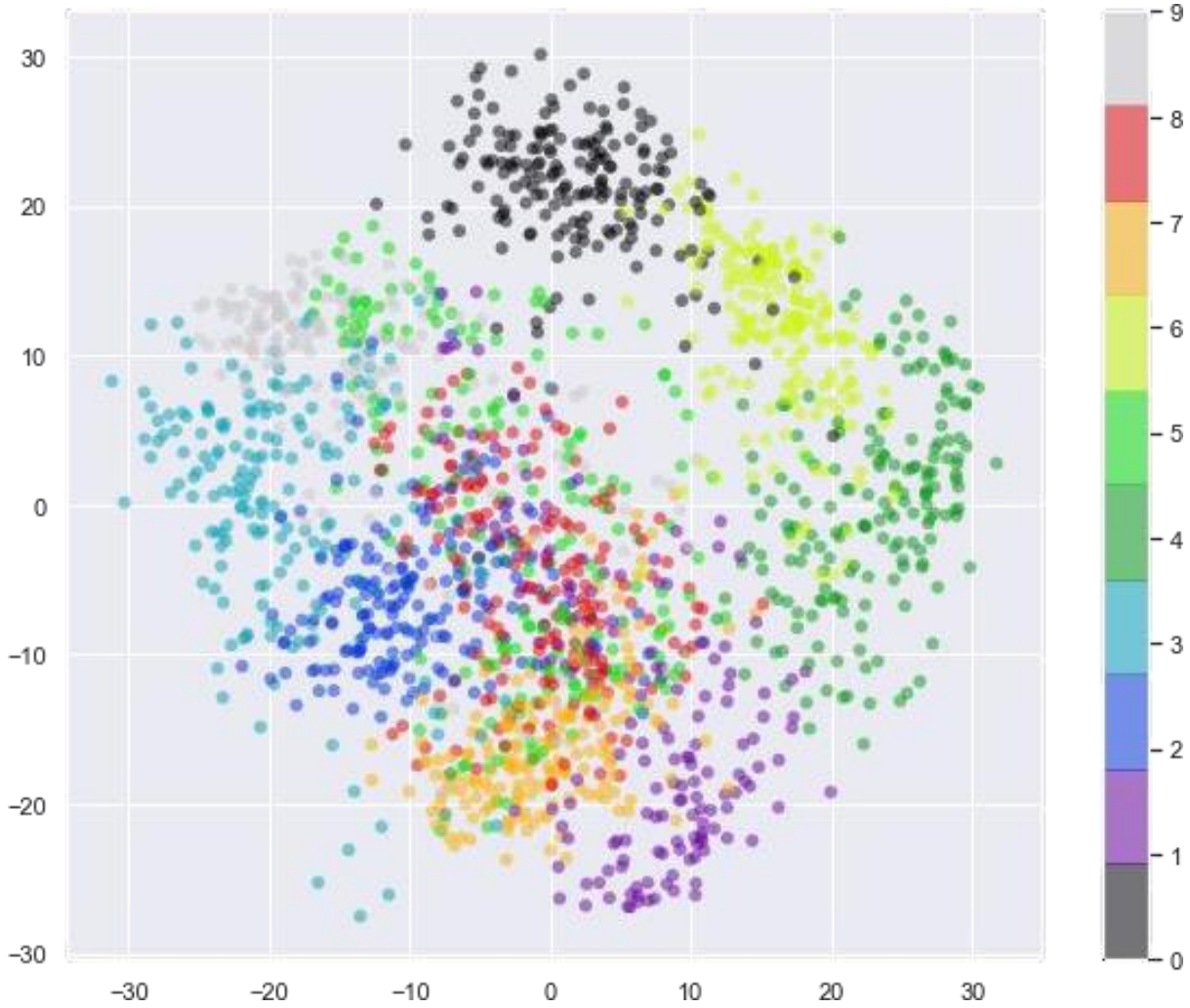
- An important question. What if we do the above process without using PCA?
  - Always do the above process first without using PCA.
  - If you don't get the desired answer, then experiment with using PCA.

Usages

Usages of PCA

- Data visualization
  - selecting the number of components: k = 2, k = 3
- Data compression
  - Reducing the memory required for data storage
  - Increasing the execution speed of the learning algorithm
  - Choosing the number of components: based on the percentage of retained variance

Usages: Data visualization



Mapping data from 64D space to 2D space In order to better visualize and understand the data

Usages: Compression

k = 100, variance = 0.91

Original Data

Usages: Compression

k = 150, variance = 0.95

Original Data

Usages: Compression

k = 200, variance = 0.97

Original Data

Usages: Compression

- Training set
  - 400 face images (gray)
- Image dimensions
  - 64 * 64 pixels
- Number of features
  - 4096 features

Usages: Compression

k = 50, variance = 0.87

Original Faces

Usages: Compression

k = 100, variance = 0.93

Original Faces

Usages: Compression

k = 150, variance = 0.96

Original Faces

Usages: Compression

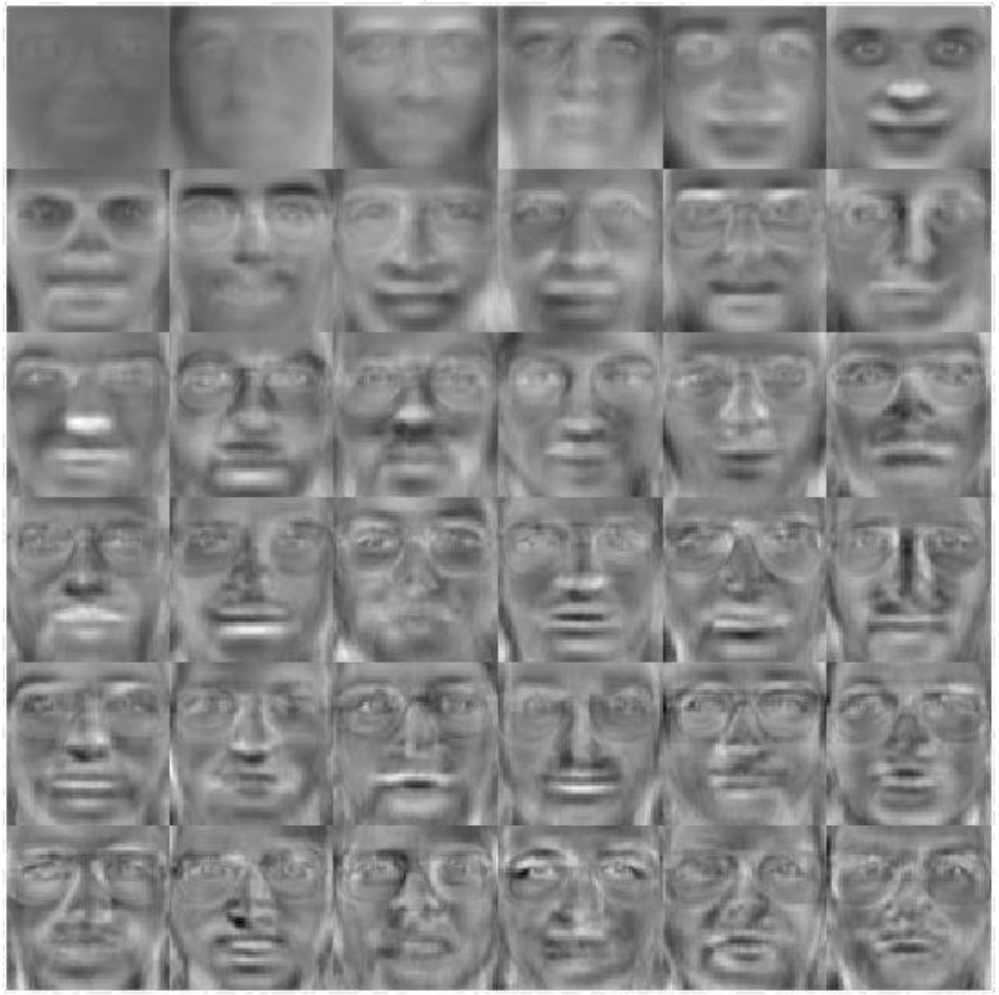
k = 200, variance = 0.98

Original Faces

Usages: Compression

k = 25, variance = 0.79

k = 36, variance = 0.84



Usages: Compression Reconstruction of images

Principal Components

k = 25, variance = 0.79

Original face

Usages: Compression Reconstruction of images

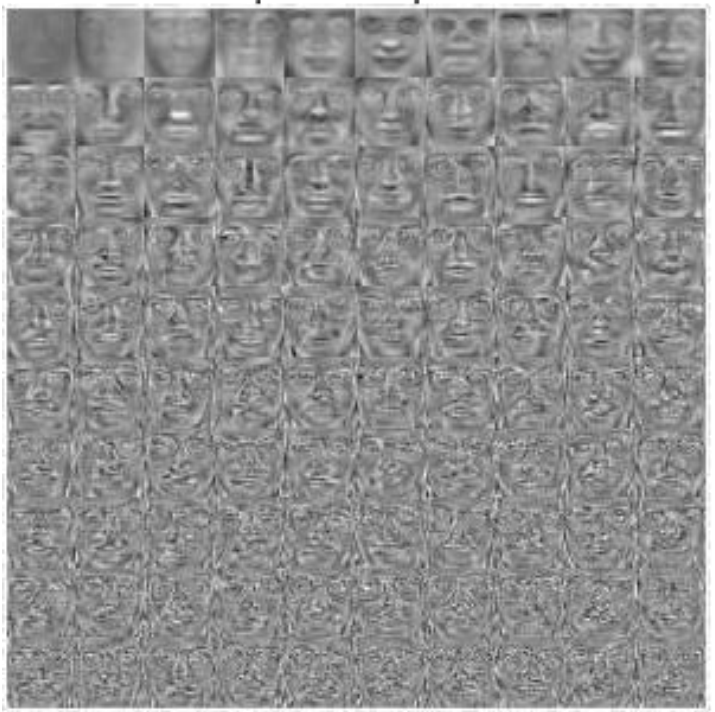
Principal Components

k = 36, variance = 0.84

Original face

Usages: Compression Reconstruction of images

Principal Components



k = 100, variance = 0.93

Original face

Appendix1: analysis of individual values

Analysis of individual values

- Motivation
  - Data simplification
  - Removal of noise and redundancy
  - Improve algorithm results
- Exemplary applications
  - Search and retrieve information
  - Recommender systems

Analysis of individual values

- analysis of individual values

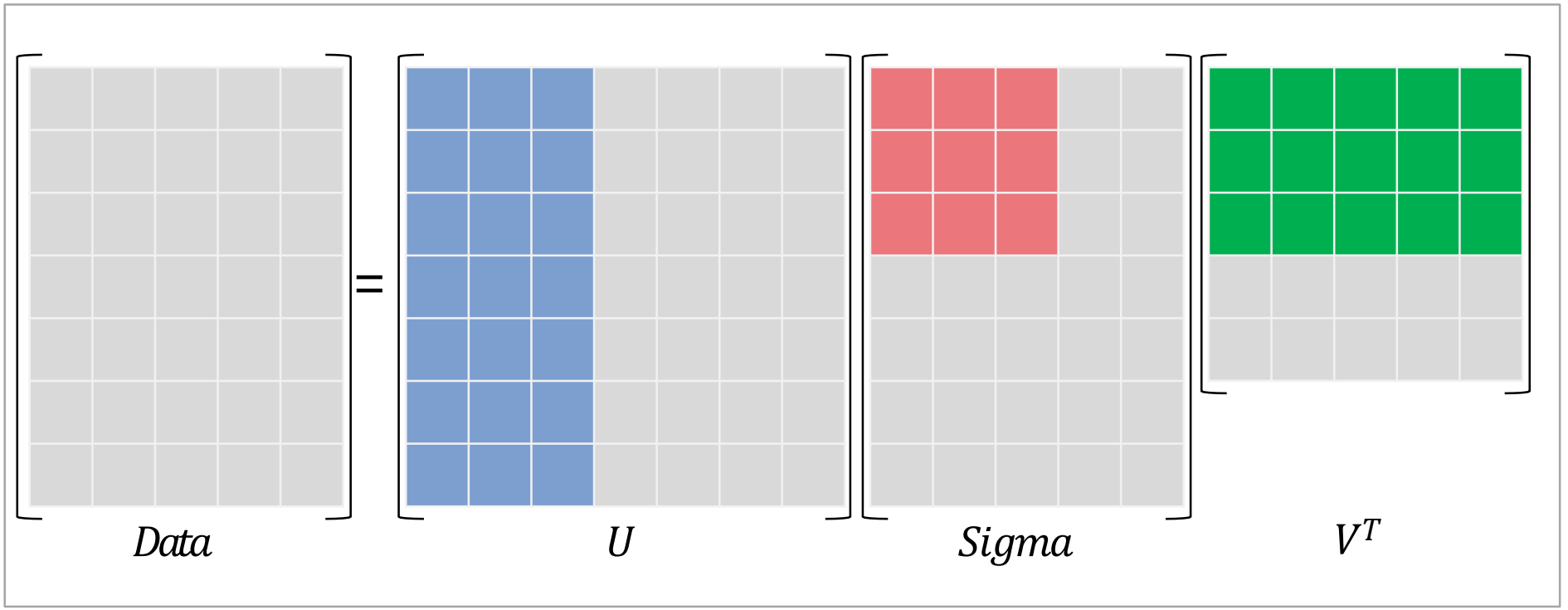
$$Data_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^{T}_{n\times n}$$

matrix of singular values

- matrix of singular values
  - A diagonal matrix in which the individual values are ordered in descending order
  - Single values from an index such as r onwards have zero value
  - The singular values of the square root are the eigenvalues of the matrix Data * Data

Analysis of individual values: example

$$Data_{m\times n} \approx U_{m\times 3}\Sigma_{3\times 3}V_{3\times n}$$



Analysis of individual values: example


```
X = np.array([[1, 1, 1, 0, 0],
              [2, 2, 2, 0, 0],
              [1, 1, 1, 0, 0],
              [5, 5, 5, 0, 0],
              [1, 1, 0, 2, 2],
              [0, 0, 0, 3, 3],
              [0, 0, 0, 1, 1]])

# Singular Value Decomposition  U,
Sigma, VT = svd(X)  print(Sigma)
```

Primary data

1	1	1	0	0
2	2	2	0	0
1	1	1	0	0
5	5	5	0	0
1	1	0	2	2
0	0	0	3	3
0	0	0	1	1

[9.72e+00 5.29e+00 6.84e-01 4.12e-16 1.36e-16]

Analysis of individual values: example


```
X_approx = U[:, :1] @ np.diag(Sigma)[:1, :1] @ VT[:1, :]

print("SSE = {:.2f}".format(np.linalg.norm(X - X_approx) ** 2))
```

$SSE \approx 28.5$

```
X_approx = U[:, :2] @ np.diag(Sigma)[:2, :2] @ VT[:2, :]

print("SSE = {:.2f}".format(np.linalg.norm(X - X_approx) ** 2))
```

$SSE \approx 0.47$

```
X_approx = U[:, :3] @ np.diag(Sigma)[:3, :3] @ VT[:3, :]

print("SSE = {:.2f}".format(np.linalg.norm(X - X_approx) ** 2))
```

$SSE \approx 0.00$

Determine the number of individual values

- Determine the number of individual values.
  - Similar to determining the number of principal components

- An experimental method. Choose the smallest K :

$$\frac{\sum_{i=1}^{k} s_i^2}{\sum_{i=1}^{n} s_i^2} \geq 0.90$$

$k = 1.$ $energy = 0.768$

$k = 2.$ $energy = 0.996$

$k = 3.$ $energy = 1.000$

9.72	**0**	**0**	**0**	**0**
0	**5.29**	**0**	**0**	**0**
0	**0**	**0.68**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**

Appendix: 2 eigenvalues and eigenvectors

Eigenvalues and Eigenvectors

- Consider the result of the square matrix A by the vector X

$$Ax = y$$

- Matrix A, like a function, transforms vector X into new vector Y,

- Eigenvector. A vector X is a special vector if it is parallel to the vector AX

$$Ax = \lambda x$$

special amount special amount

- Special amount. In the above relation, λ is the eigenvalue corresponding to the eigenvector X.

Eigenvalues and Eigenvectors

- Example: If A is a permutation matrix as follows:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- in this case

$$A\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \lambda = 1$$

Orthogonal

$$A\begin{bmatrix} -1 \\ 1 \end{bmatrix} = (-1) \times \begin{bmatrix} -1 \\ 1 \end{bmatrix} \Rightarrow \lambda = -1$$

$$\text{trace}(A) = 0 + 0 = 1 + (-1)$$

- Attention. The sum of the eigenvalues is equal to the sum of the elements of the main diameter

Eigenvalues and Eigenvectors

- Calculation of eigenvalues.

$$Ax = \lambda x \Rightarrow (A - \lambda I)\, x = 0$$

- Therefore, the matrix λ $-$ is a unique matrix. [because the vector is empty]

- As a result:

$$det\,(A - \lambda I) = 0$$

- Example: Calculation of eigenvalues.

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

trace(A) det(A)

$$\text{Det}(A - \lambda I) = \det\left(\begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix}\right) = \lambda^2 - 6\lambda + 8 = 0 \Rightarrow \lambda = 4, 2$$

Eigenvalues and Eigenvectors

- Calculating Eigenvalues
- Example:

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \Rightarrow \lambda_1 = 4,\ \lambda_2 = 2$$

$$(A - 4I)\, x_1 = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} x_1 = 0 \Rightarrow x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$(A - 2I)\, x_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} x_2 = 0 \Rightarrow x_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

- result

$$Ax = \lambda x \Rightarrow \quad (A + 3I)\, x = Ax + 3x = \lambda x + 3x = \quad (\lambda + 3)\, x$$

Matrix decomposition A: diagonalization

- Assume that S is a matrix whose columns are the eigenvectors of A matrix.

$$
\begin{aligned}
AS &= A\begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \\
&= \begin{bmatrix} | & | & \cdots & | \\ \lambda_1 x_1 & \lambda_2 x_2 & \cdots & \lambda_n x_n \\ | & | & \cdots & | \end{bmatrix} \\
&= \begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \\
&= S\Lambda
\end{aligned}
$$

$$AS = S\Lambda \Rightarrow S^{-1}AS = \Lambda$$

$$AS = S\Lambda \Rightarrow A = S\Lambda S^{-1}$$

Decomposition of Matrix A: Diagonalization

 view. If we raise the matrix A to the power of two, the eigenvalues are raised to the power of two and the eigenvectors They do not change.

$$A = S\Lambda S^{-1} \Rightarrow A^2 = S\Lambda S^{-1} S\Lambda S^{-1} = S\Lambda^2 S^{-1}$$

 In general.

$$A = S\Lambda S^{-1} \Rightarrow A^k = S\Lambda^k S^{-1}$$

Symmetric Matrix Decomposition

- In a real symmetric matrix:
  - The eigenvalues are real.
  - The orthogonal eigenvectors are normal.

$$A = Q \Lambda Q^{-1} = Q \Lambda Q^{T}$$

$$A = \begin{bmatrix} | & | & \cdots & | \\ q_1 & q_2 & \cdots & q_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} - & q_1^T & - \\ - & q_2^T & - \\ \vdots & \vdots & \vdots \\ - & q_n^T & - \end{bmatrix}$$

$$= \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T$$

- view. Any symmetric linear combination matrix is a set of orthogonal projection matrices.

Symmetric Positive Definite Matrices

- Positive definite matrix. The matrix A is positive definite if for every non-zero vector such as x:

$$x^T Ax > 0$$

- In a symmetric positive definite matrix:
  - The eigenvalues are all positive.
  - The axes are all positive.
  - All determinants (determinants of the preceding submatrices) are positive.

- Example.

$$A = \begin{bmatrix} 5 & 2 \\ 2 & 3 \end{bmatrix} \Rightarrow \lambda = 4 \pm \sqrt{5}, p_1 = 5, p_2 = \frac{11}{5}, \det(A) = 11$$