

Mathematics for Artificial Intelligence

2강: 행렬은 뭔가요?



임성빈 **UNIST** 인공지능대학원 & 산업공학과
Learning Intelligent Machine Lab

행렬은 뭔가요?

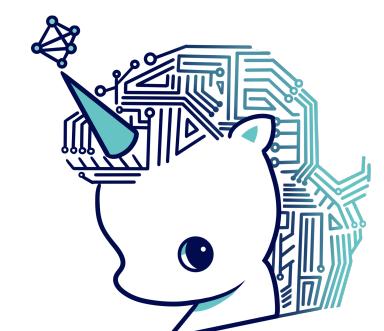
- 행렬(matrix)은 벡터를 원소로 가지는 2차원 배열입니다

수식

$$X = \begin{bmatrix} 1 & -2 & 3 \\ 7 & 5 & 0 \\ -2 & -1 & 2 \end{bmatrix}$$

코드

```
1 X = np.array([[1,-2, 3],  
2                      [7, 5, 0],  
3                      [-2, -1, 2]])
```



numpy 에선 행(row)이 기본 단위입니다

행렬은 뭔가요?

- 행렬(matrix)은 벡터를 원소로 가지는 2차원 배열입니다

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & y_{2m} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_n$$

$n \times m$ 행렬

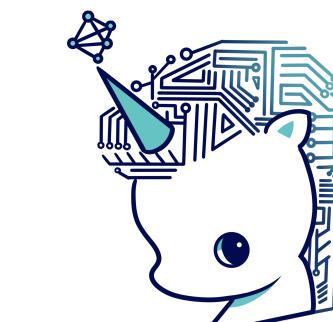
X

행렬은 뭔가요?

- 행렬(matrix)은 벡터를 원소로 가지는 2차원 배열입니다
- 행렬은 행(row)과 열(column)이라는 인덱스(index)를 가집니다

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & y_{2m} \\ \vdots & \vdots & x_{ij} & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{matrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{matrix}$$

$n \times m$ 행렬



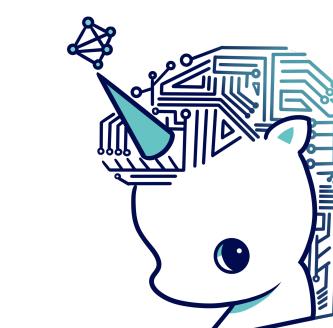
$\mathbf{X} = (x_{ij})$
위와 같이 표기하기도 합니다

행렬은 뭔가요?

- 행렬(matrix)은 벡터를 원소로 가지는 2차원 배열입니다
- 행렬은 행(row)과 열(column)이라는 인덱스(index)를 가집니다
- 행렬의 특정 행(열)을 고정하면 행(열)벡터라 부릅니다

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & y_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{array}{l} \mathbf{x}_1 \text{ 행벡터} \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{array}$$

열벡터 $n \times m$ 행렬



$\mathbf{X} = (x_{ij})$
위와 같이 표기하기도 합니다

행렬은 뭔가요?

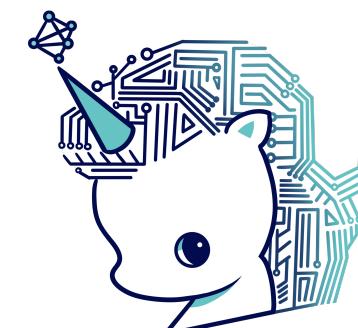
- 행렬(matrix)은 벡터를 원소로 가지는 2차원 배열입니다
 - 행렬은 행(row)과 열(column)이라는 인덱스(index)를 가집니다
 - 행렬의 특정 행(열)을 고정하면 행(열)벡터라 부릅니다

$$\mathbf{X}^\top = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1m} & x_{2m} & \cdots & x_{nm} \end{bmatrix}$$

행벡터

열벡터

$m \times n$ 행렬

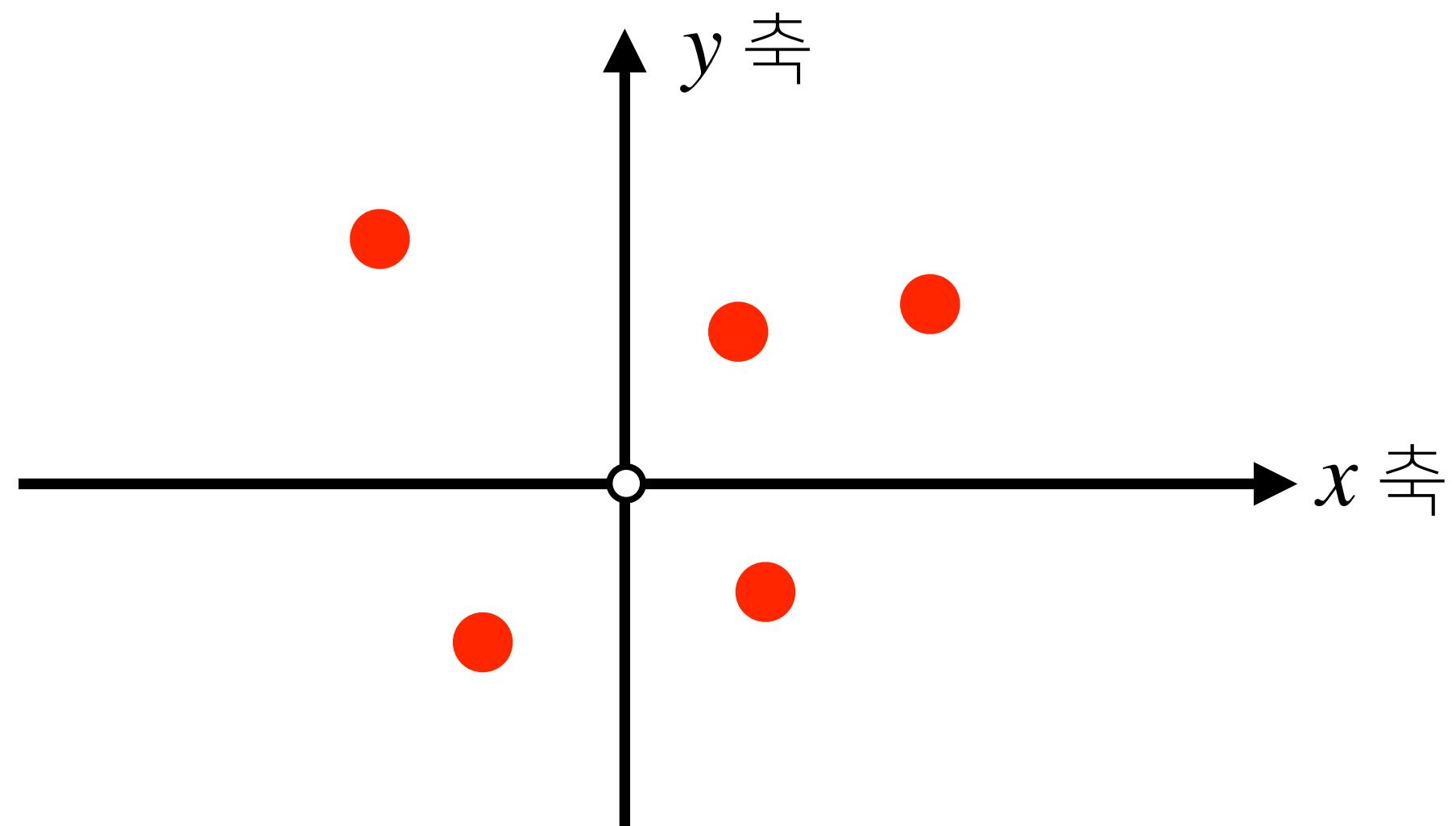


$$X^\top = (x_{ji})$$

전치행렬(transpose matrix)은 행과 열의 인덱스가 바뀐 행렬을 말합니다

행렬을 이해하는 방법 (1)

- 벡터가 공간에서 한 점을 의미한다면 행렬은 여러 점들을 나타냅니다



2차원 공간 (좌표평면)

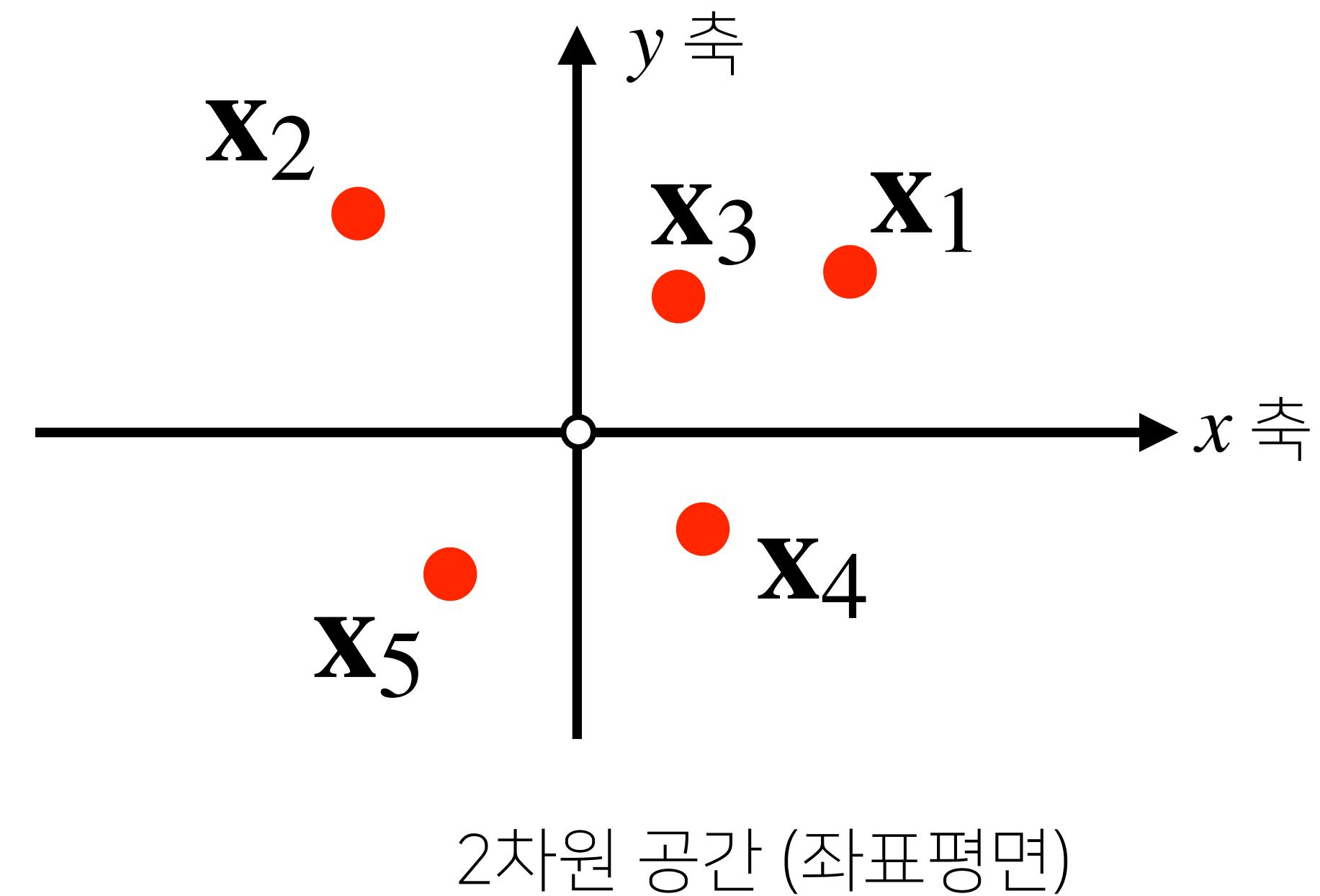
X

행렬을 이해하는 방법 (1)

- 벡터가 공간에서 한 점을 의미한다면 행렬은 여러 점들을 나타냅니다
- 행렬의 행벡터 \mathbf{x}_i 는 *i* 번째 데이터를 의미합니다

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{array}$$

×

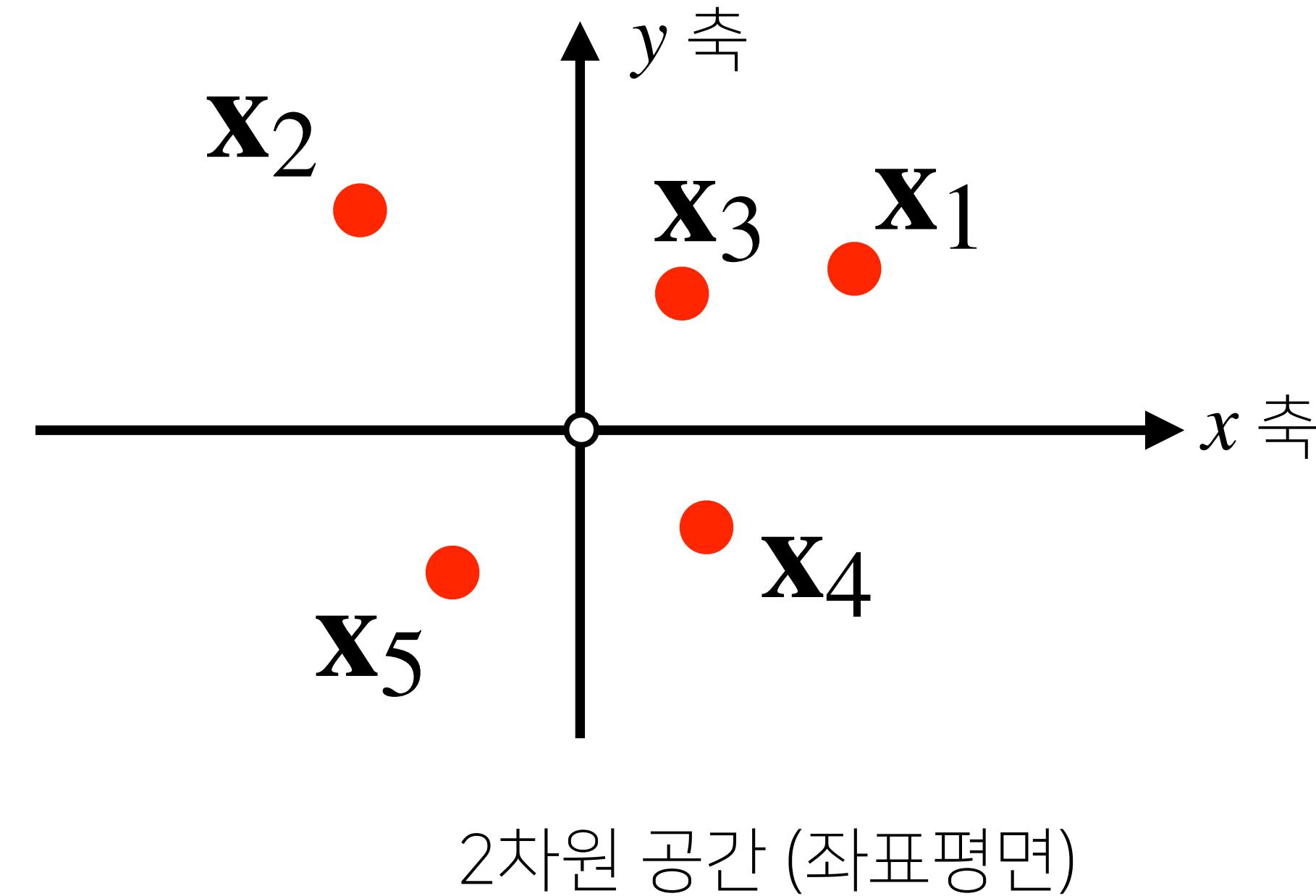


행렬을 이해하는 방법 (1)

- 벡터가 공간에서 한 점을 의미한다면 행렬은 여러 점들을 나타냅니다
- 행렬의 행벡터 \mathbf{x}_i 는 i 번째 데이터를 의미합니다
- 행렬의 x_{ij} 는 i 번째 데이터의 j 번째 변수의 값을 말합니다

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{array}{l} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{array}$$

X

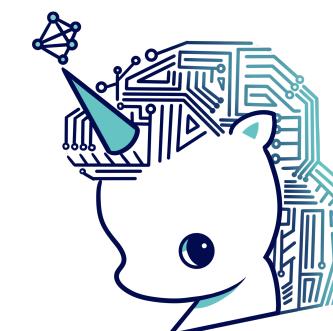


행렬의 덧셈, 뺄셈, 성분곱, 스칼라곱

- 행렬은 벡터를 원소로 가지는 2차원 배열입니다
- 행렬끼리 같은 모양을 가지면 덧셈, 뺄셈을 계산할 수 있습니다

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}$$

$$\mathbf{X} \pm \mathbf{Y} = (x_{ij} \pm y_{ij})$$



벡터의 덧뺄셈과 다른게 없습니다

$$\mathbf{X} \pm \mathbf{Y} = \begin{bmatrix} x_{11} \pm y_{11} & x_{12} \pm y_{12} & \cdots & x_{1m} \pm y_{1m} \\ x_{21} \pm y_{21} & x_{22} \pm y_{22} & \cdots & x_{2m} \pm y_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} \pm y_{n1} & x_{n2} \pm y_{n2} & \cdots & x_{nm} \pm y_{nm} \end{bmatrix}$$

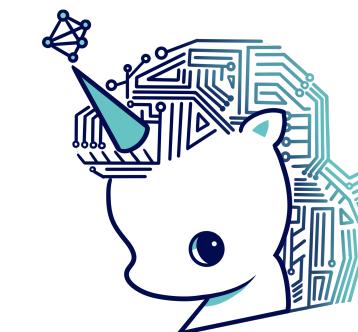
X

행렬의 덧셈, 뺄셈, 성분곱, 스칼라곱

- 행렬은 벡터를 원소로 가지는 2차원 배열입니다
- 행렬끼리 같은 모양을 가지면 덧셈, 뺄셈을 계산할 수 있습니다
- 성분곱은 벡터와 똑같습니다.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}$$

$$\mathbf{X} \odot \mathbf{Y} = (x_{ij} y_{ij})$$



성분곱은 각 인덱스 위치끼리 곱합니다

$$\mathbf{X} \odot \mathbf{Y} = \begin{bmatrix} x_{11}y_{11} & x_{12}y_{12} & \cdots & x_{1m}y_{1m} \\ x_{21}y_{21} & x_{22}y_{22} & \cdots & x_{2m}y_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1}y_{n1} & x_{n2}y_{n2} & \cdots & x_{nm}y_{nm} \end{bmatrix}$$

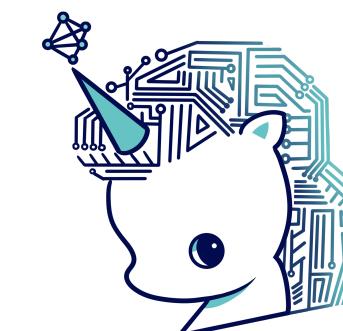
X

행렬의 덧셈, 뺄셈, 성분곱, 스칼라곱

- 행렬은 벡터를 원소로 가지는 2차원 배열입니다
- 행렬끼리 같은 모양을 가지면 덧셈, 뺄셈을 계산할 수 있습니다
- 성분곱은 벡터와 똑같습니다. 스칼라곱도 벡터와 차이가 없습니다

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}$$

×



모든 성분에 똑같이 숫자를 곱해줍니다

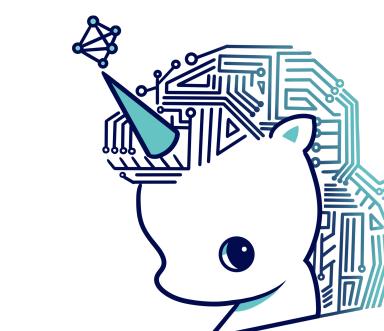
$$\alpha\mathbf{X} = \begin{bmatrix} \alpha x_{11} & \alpha x_{12} & \cdots & \alpha x_{1m} \\ \alpha x_{21} & \alpha x_{22} & \cdots & \alpha x_{2m} \\ \vdots & \vdots & & \vdots \\ \alpha x_{n1} & \alpha x_{n2} & \cdots & \alpha x_{nm} \end{bmatrix}$$

행렬 곱셈

- 행렬 곱셈(matrix multiplication)은 i 번째 행벡터와 j 번째 열벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1\ell} \\ y_{21} & y_{22} & \cdots & y_{2\ell} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{m\ell} \end{bmatrix}$$

$$\mathbf{XY} = \left(\sum_k x_{ik} y_{kj} \right)$$



행렬곱은 \mathbf{X} 의 열의 개수와 \mathbf{Y} 의 행의 개수가 같아야 한다

\times

행렬 곱셈

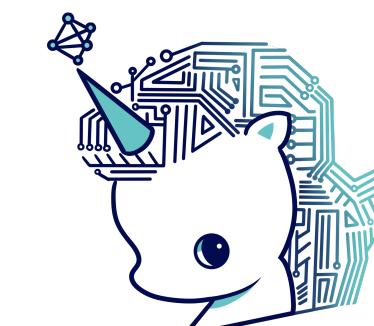
- 행렬 곱셈(matrix multiplication)은 i 번째 행벡터와 j 번째 열벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다

numpy 에선 `@` 연산을 사용한다

```
1 X = np.array([[1, -2, 3],  
2                 [7, 5, 0],  
3                 [-2, -1, 2]])  
4  
5 Y = np.array([[0, 1],  
6                 [1, -1],  
7                 [-2, 1]])
```

```
1 X @ Y  
  
array([[ -8,   6],  
      [  5,   2],  
      [-5,   1]])
```

$$XY = \left(\sum_k x_{ik} y_{kj} \right)$$



$$1 \cdot 0 + (-2) \cdot 1 + 3 \cdot (-2) = -8$$

X

행렬 곱셈

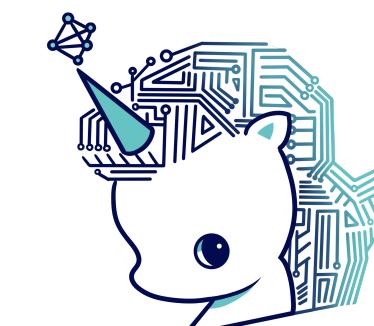
- 행렬 곱셈(matrix multiplication)은 i 번째 행벡터와 j 번째 열벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다

numpy 에선 `@` 연산을 사용한다

```
1 X = np.array([[1, -2, 3],  
2                 [7, 5, 0],  
3                 [-2, -1, 2]])  
4  
5 Y = np.array([[0, 1],  
6                 [1, -1],  
7                 [-2, 1]])
```

```
1 X @ Y  
  
array([[-8,  6],  
      [ 5,  2],  
      [-5,  1]])
```

$$XY = \left(\sum_k x_{ik} y_{kj} \right)$$



$$1 \cdot 1 + (-2) \cdot (-1) + 3 \cdot 1 = 6$$

\times

행렬 곱셈

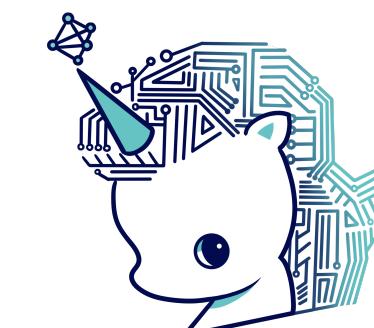
- 행렬 곱셈(matrix multiplication)은 i 번째 행벡터와 j 번째 열벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다

numpy 에선 @ 연산을 사용한다

```
1 X = np.array([[1, -2, 3],  
2                 [7, 5, 0],  
3                 [-2, -1, 2]])  
4  
5 Y = np.array([[0, 1],  
6                 [1, -1],  
7                 [-2, 1]])
```

```
1 X @ Y  
  
array([[-8,  6],  
      [ 5,  2],  
     [-5,  1]])
```

$$XY = \left(\sum_k x_{ik} y_{kj} \right)$$



$$7 \cdot 0 + 5 \cdot 1 + 0 \cdot (-2) = 5$$

X

행렬 곱셈

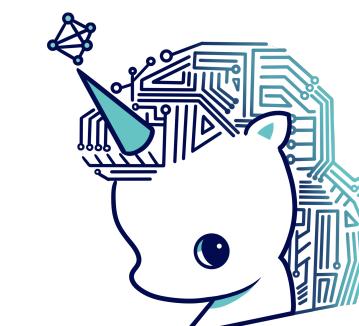
- 행렬 곱셈(matrix multiplication)은 i 번째 행벡터와 j 번째 열벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다

numpy 에선 @ 연산을 사용한다

```
1 X = np.array([[1, -2, 3],  
2                 [7, 5, 0],  
3                 [-2, -1, 2]])  
4  
5 Y = np.array([[0, 1],  
6                 [1, -1],  
7                 [-2, 1]])
```

```
1 X @ Y  
  
array([[-8,  6],  
      [ 5,  2],  
     [-5,  1]])
```

$$XY = \left(\sum_k x_{ik} y_{kj} \right)$$

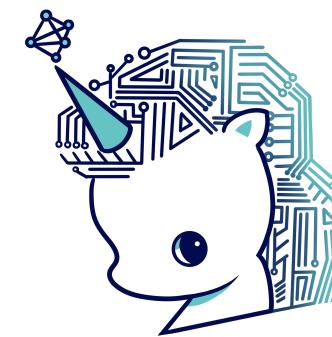


$$7 \cdot 1 + 5 \cdot (-1) + 0 \cdot 1 = 2$$

X

행렬도 내적이 있을까?

- 넘파이의 `np.inner` 는 i 번째 행벡터와 j 번째 행벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다
- 수학에서 말하는 내적과는 다르므로 주의!



수학에선 보통 $\text{tr}(\mathbf{XY}^\top)$ 을 내적으로 계산한다

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}$$
$$\mathbf{XY}^\top = \left(\sum_k x_{ik} y_{jk} \right)$$

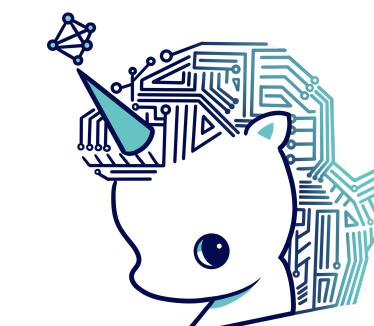
행렬도 내적이 있을까?

- 넘파이의 `np.inner` 는 i 번째 행벡터와 j 번째 행벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다
- 수학에서 말하는 내적과는 다르므로 주의!

```
1 X = np.array([[1 , -2,  3],  
2                 [7,  5 ,0],  
3                 [-2, -1 ,2]])  
4  
5 Y = np.array([[0 , 1, -1],  
6                 [1, -1 ,0]])
```

```
1 np.inner(X, Y)  
  
array([[-5,  3],  
       [ 5,  2],  
      [-3, -1]])
```

$$XY^\top = \left(\sum_k x_{ik} y_{jk} \right)$$



$$1 \cdot 0 + (-2) \cdot 1 + 3 \cdot (-1) = -5$$

X

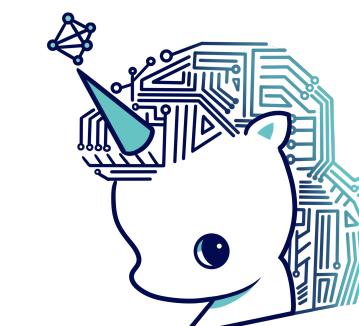
행렬도 내적이 있을까?

- 넘파이의 `np.inner` 는 i 번째 행벡터와 j 번째 행벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다
- 수학에서 말하는 내적과는 다르므로 주의!

```
1 X = np.array([[1, -2, 3],  
2                 [7, 5, 0],  
3                 [-2, -1, 2]])  
4  
5 Y = np.array([[0, 1, -1],  
6                 [1, -1, 0]])
```

```
1 np.inner(X, Y)  
  
array([[-5, 3],  
       [ 5, 2],  
       [-3, -1]])
```

$$XY^\top = \left(\sum_k x_{ik} y_{jk} \right)$$



$$1 \cdot 1 + (-2) \cdot (-1) + 3 \cdot 0 = 3$$

X

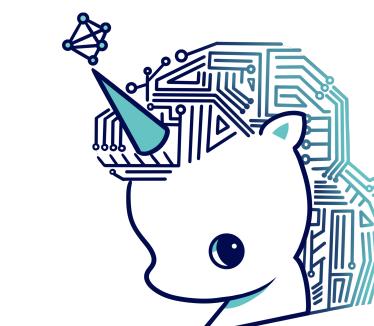
행렬도 내적이 있을까?

- 넘파이의 `np.inner` 는 i 번째 행벡터와 j 번째 행벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다
- 수학에서 말하는 내적과는 다르므로 주의!

```
1 X = np.array([[1 , -2,  3],  
2                 [7,  5 ,0],  
3                 [-2, -1 ,2]])  
4  
5 Y = np.array([[0 , 1, -1],  
6                 [1, -1 ,0]])
```

```
1 np.inner(X, Y)  
  
array([[-5,  3],  
       [ 5,  2],  
       [-3, -1]])
```

$$XY^\top = \left(\sum_k x_{ik} y_{jk} \right)$$



$$7 \cdot 0 + 5 \cdot 1 + 0 \cdot (-1) = 5$$

X

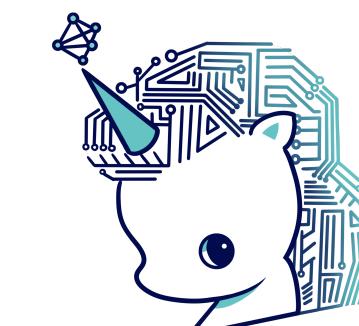
행렬도 내적이 있을까?

- 넘파이의 `np.inner` 는 i 번째 행벡터와 j 번째 행벡터 사이의 내적을 성분으로 가지는 행렬을 계산합니다
- 수학에서 말하는 내적과는 다르므로 주의!

```
1 X = np.array([[1 , -2,  3],  
2                 [7,  5 ,0],  
3                 [-2, -1 ,2]])  
4  
5 Y = np.array([[0 , 1, -1],  
6                 [1, -1 ,0]])
```

```
1 np.inner(X, Y)  
  
array([[-5,  3],  
       [ 5,  2],  
      [-3, -1]])
```

$$XY^\top = \left(\sum_k x_{ik} y_{jk} \right)$$



$$7 \cdot 1 + 5 \cdot (-1) + 0 \cdot 0 = 2$$

X

행렬을 이해하는 방법 (2)

- 행렬은 벡터공간에서 사용되는 연산자(operator)로 이해한다

$$z_i = \sum_j a_{ij} x_j$$
$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

z \times **A** **x**

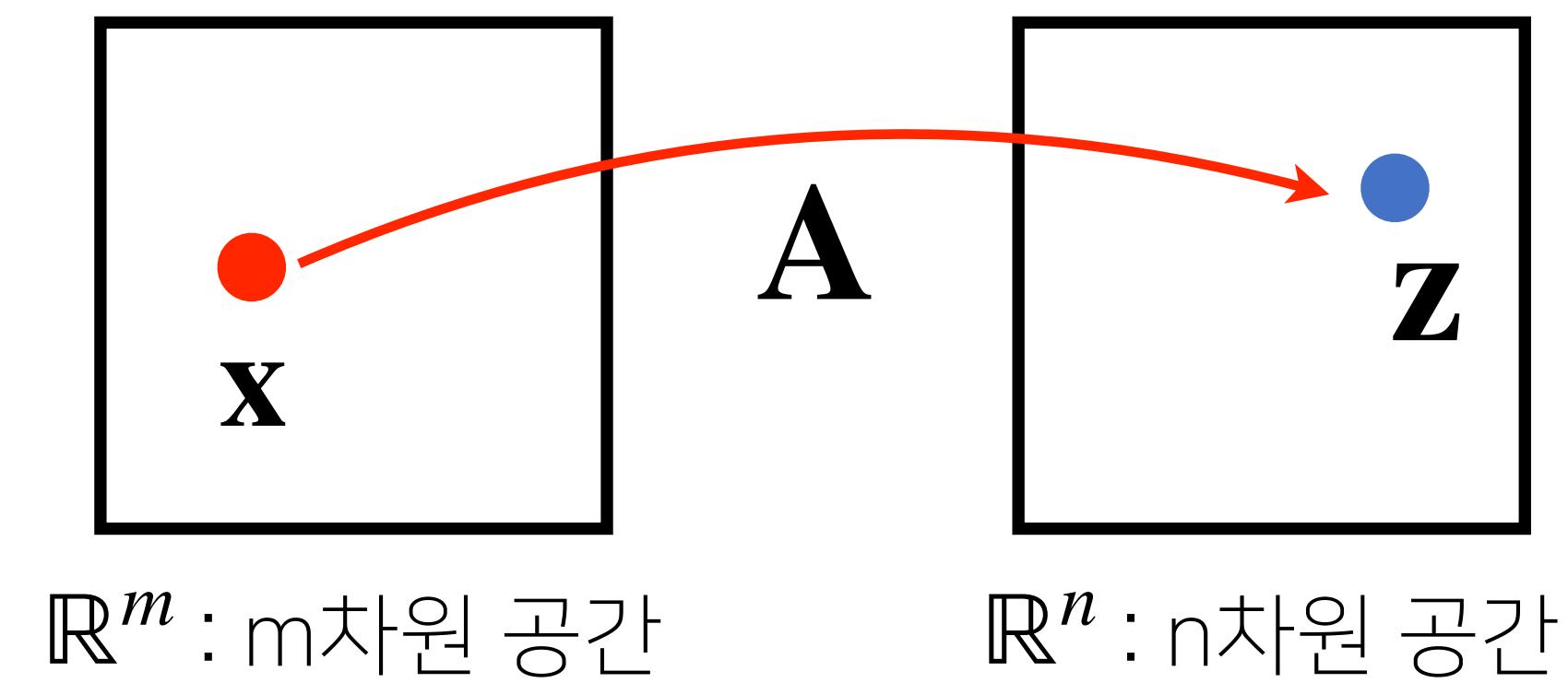
행렬을 이해하는 방법 (2)

- 행렬은 벡터공간에서 사용되는 연산자(operator)로 이해한다
- 행렬곱을 통해 벡터를 다른 차원의 공간으로 보낼 수 있습니다

$$z_i = \sum_j a_{ij} x_j$$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

z \times **A** **x**

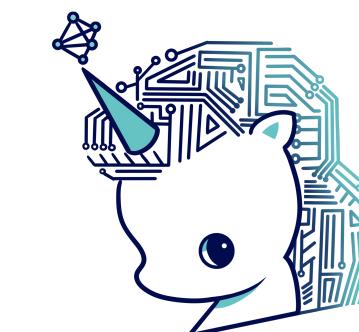


행렬을 이해하는 방법 (2)

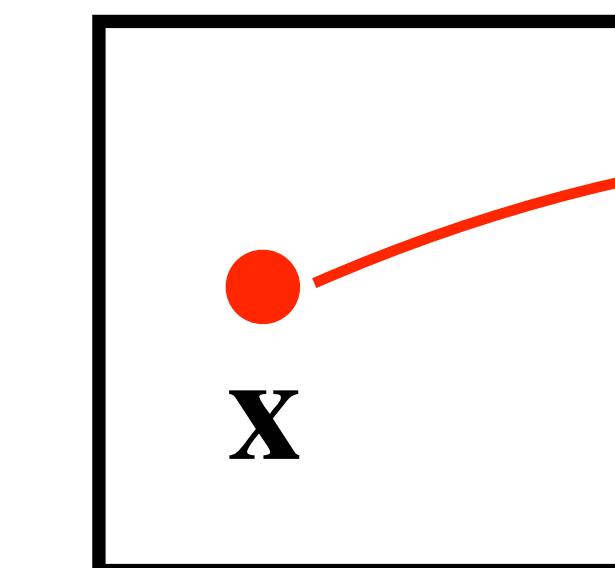
- 행렬은 벡터공간에서 사용되는 연산자(operator)로 이해한다
- 행렬곱을 통해 벡터를 다른 차원의 공간으로 보낼 수 있습니다
- 행렬곱을 통해 패턴을 추출할 수 있고 데이터를 압축할 수도 있습니다

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

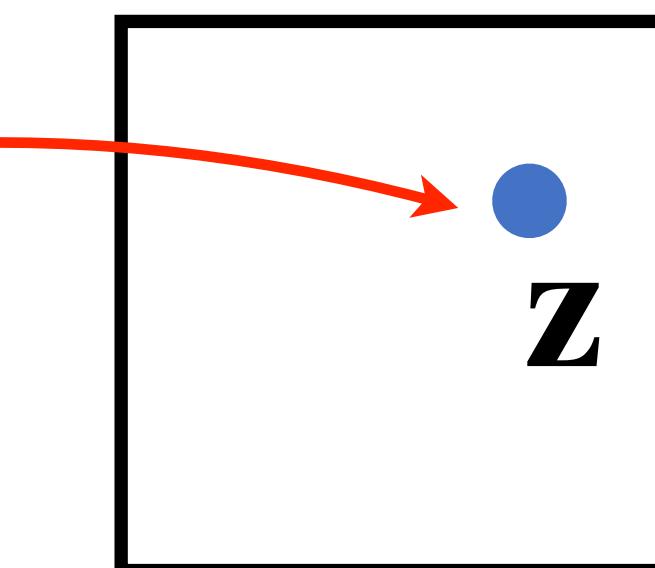
A



모든 선형변환(linear transform)은
행렬곱으로 계산할 수 있다



\mathbb{R}^m : m차원 공간



\mathbb{R}^n : n차원 공간

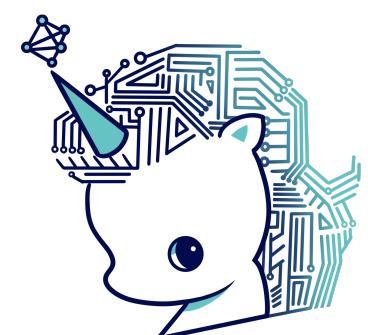
역행렬 이해하기

- 어떤 행렬 A 의 연산을 거꾸로 되돌리는 행렬을 역행렬(inverse matrix)이라 부르고 A^{-1} 라 표기한다. 역행렬은 행과 열 숫자가 같고 행렬식(determinant)이 0이 아닌 경우에만 계산할 수 있다

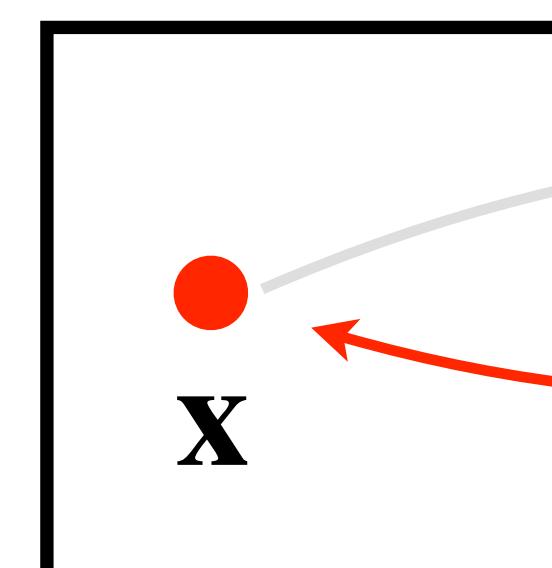
$$AA^{-1} = A^{-1}A = I$$

항등행렬

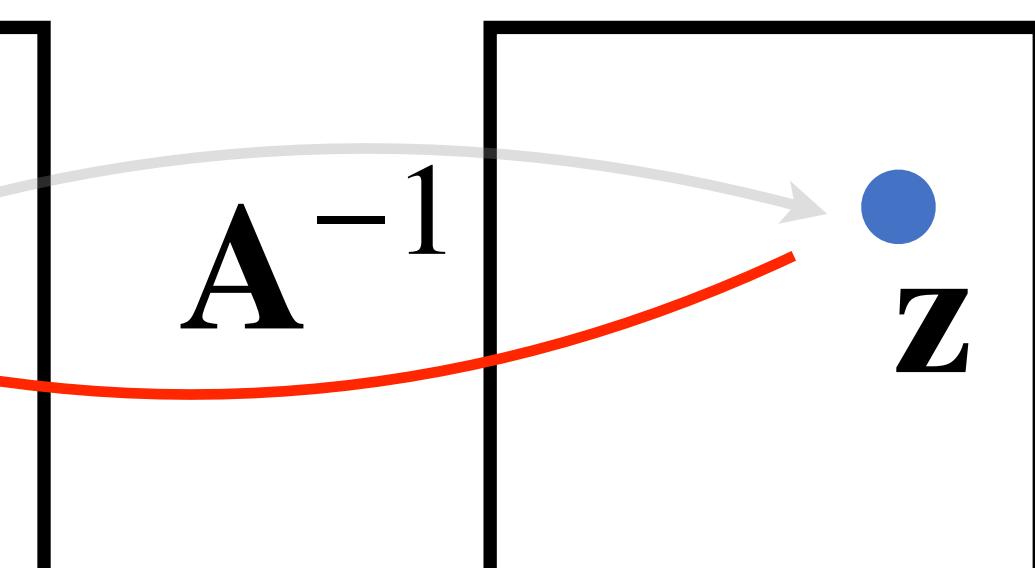
$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$



역행렬 연산은 $n = m$ 일 때만 가능하고
행렬 A 의 행렬식이 0 이 되면 안된다



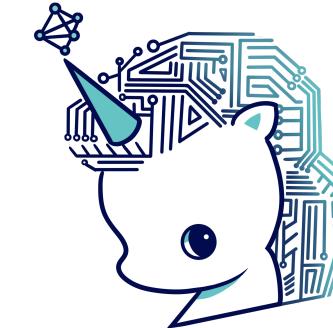
\mathbb{R}^n : n차원 공간



\mathbb{R}^n : n차원 공간

X

역행렬 이해하기



`numpy.linalg.inv`로 구할 수 있다

- 어떤 행렬 A 를 라부르고 A 의 역행렬 (determinant)

A

```
1 X = np.array([[1,-2, 3],  
2                 [7, 5, 0],  
3                 [-2, -1, 2]])
```

```
1 np.linalg.inv(X)
```

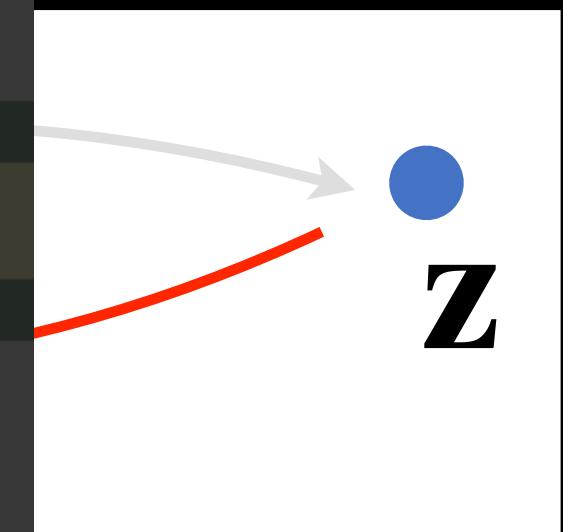
```
array([[ 0.21276596,  0.0212766 , -0.31914894],  
       [-0.29787234,  0.17021277,  0.44680851],  
       [ 0.06382979,  0.10638298,  0.40425532]])
```

```
1 X @ np.linalg.inv(X)
```

```
array([[ 1.0000000e+00, -1.38777878e-17,  0.0000000e+00],  
       [-2.22044605e-16,  1.0000000e+00, -5.55111512e-17],  
       [-2.77555756e-17,  0.0000000e+00,  1.0000000e+00]])
```

$\rightarrow \text{matrix}()$ 이
식

: m 일 때만 가능하고
0 이 되면 안된다



\mathbb{R}^n : n차원 공간

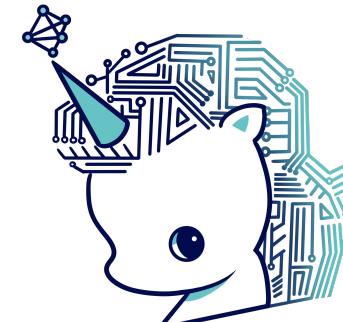
\times

역행렬 이해하기

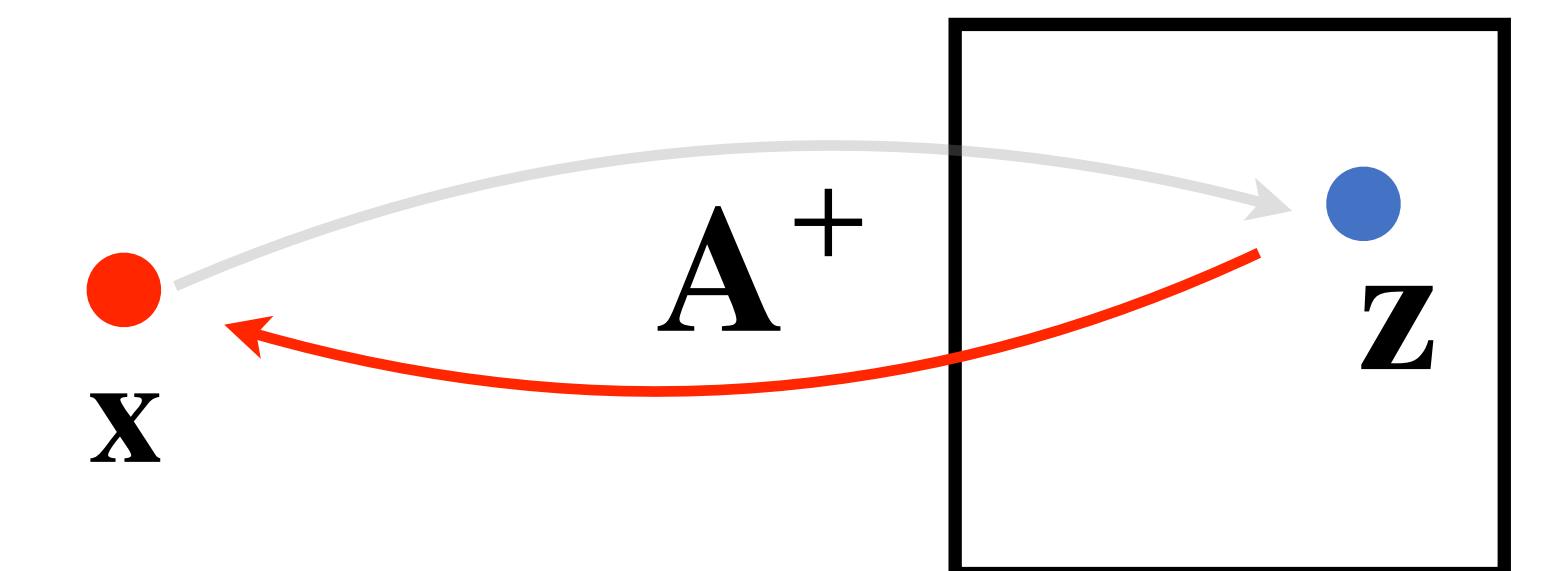
- 어떤 행렬 A 의 연산을 거꾸로 되돌리는 행렬을 역행렬(inverse matrix)이라 부르고 A^{-1} 라 표기한다. 역행렬은 행과 열 숫자가 같고 행렬식(determinant)이 0이 아닌 경우에만 계산할 수 있다
- 만일 역행렬을 계산할 수 없다면 유사역행렬(pseudo-inverse) 또는 무어-펜로즈(Moore-Penrose) 역행렬 A^+ 을 이용한다

$$n \geq m \text{ 인 경우 } A^+ = (A^\top A)^{-1} A^\top$$

$$n \leq m \text{ 인 경우 } A^+ = A^\top (A A^\top)^{-1}$$

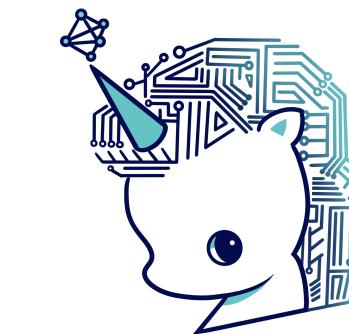


$n \geq m$ 이면 $A^+ A = I$ 가 성립하고
 $n \leq m$ 이면 $A A^+ = I$ 만 성립한다



\mathbb{R}^m : m차원 공간

\mathbb{R}^n : n차원 공간



`numpy.linalg.pinv`로 구할 수 있다

역행렬 이해하기

- 어떤 행
라 부르
(determ)
- 만일 역
펜로즈(

$n \geq m$ 인 경우

$n \leq m$ 인 경우

```
1 Y = np.array([[0 ,1 ,  
                 [1, -1],  
                 [-2, 1]]])  
  
1 np.linalg.pinv(Y)  
  
array([[ 5.0000000e-01,  4.09730229e-17, -5.0000000e-01],  
       [ 8.3333333e-01, -3.3333333e-01, -1.6666667e-01]])  
  
1 np.linalg.pinv(Y) @ Y  
  
array([[ 1.0000000e+00, -2.22044605e-16],  
       [ 0.0000000e+00,  1.0000000e+00]])
```

matrix)이

= 무어-

z

차원 공간

X

응용1: 연립방정식 풀기

- `np.linalg.pinv` 를 이용하면 연립방정식의 해를 구할 수 있다

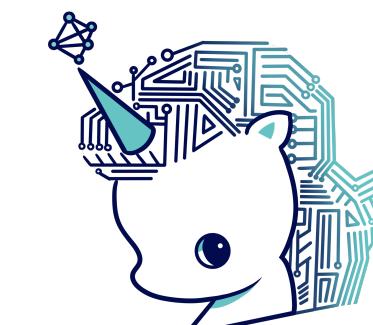
$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1$$

$$a_{12}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2$$

⋮
⋮
⋮

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n$$

$n \leq m$ 인 경우: 식이 변수 개수보다 작거나 같아야 함



(a_{ij}) 와 (b_i) 들이 주어진 상황에서
방정식을 만족하는 (x_j) 를 구하는 상황이다

×

응용1: 연립방정식 풀기

- `np.linalg.pinv` 를 이용하면 연립방정식의 해를 구할 수 있다

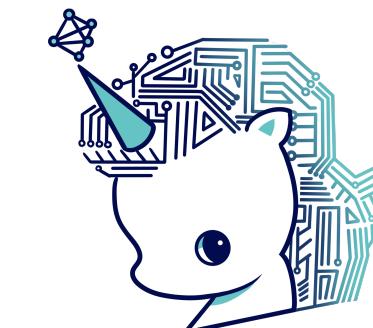
$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1$$

$$a_{12}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2$$

⋮
⋮
⋮

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n$$

$n \leq m$ 인 경우: 식이 변수 개수보다 작거나 같아야 함



연립방정식은 행렬을 사용하면
아래와 같이 표현할 수 있다

$$\rightarrow \mathbf{Ax} = \mathbf{b}$$

×

응용1: 연립방정식 풀기

- `np.linalg.pinv` 를 이용하면 연립방정식의 해를 구할 수 있다

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1$$

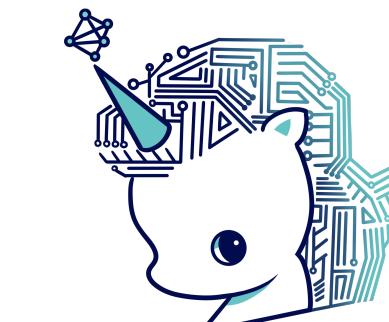
$$a_{12}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2$$

⋮

⋮

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n$$

$n \leq m$ 인 경우: 식이 변수 개수보다 작거나 같아야 함



$n \leq m$ 이면 무어-펜로즈 역행렬을 이용하면 해를 하나 구할 수 있다

$$\rightarrow \mathbf{Ax} = \mathbf{b}$$

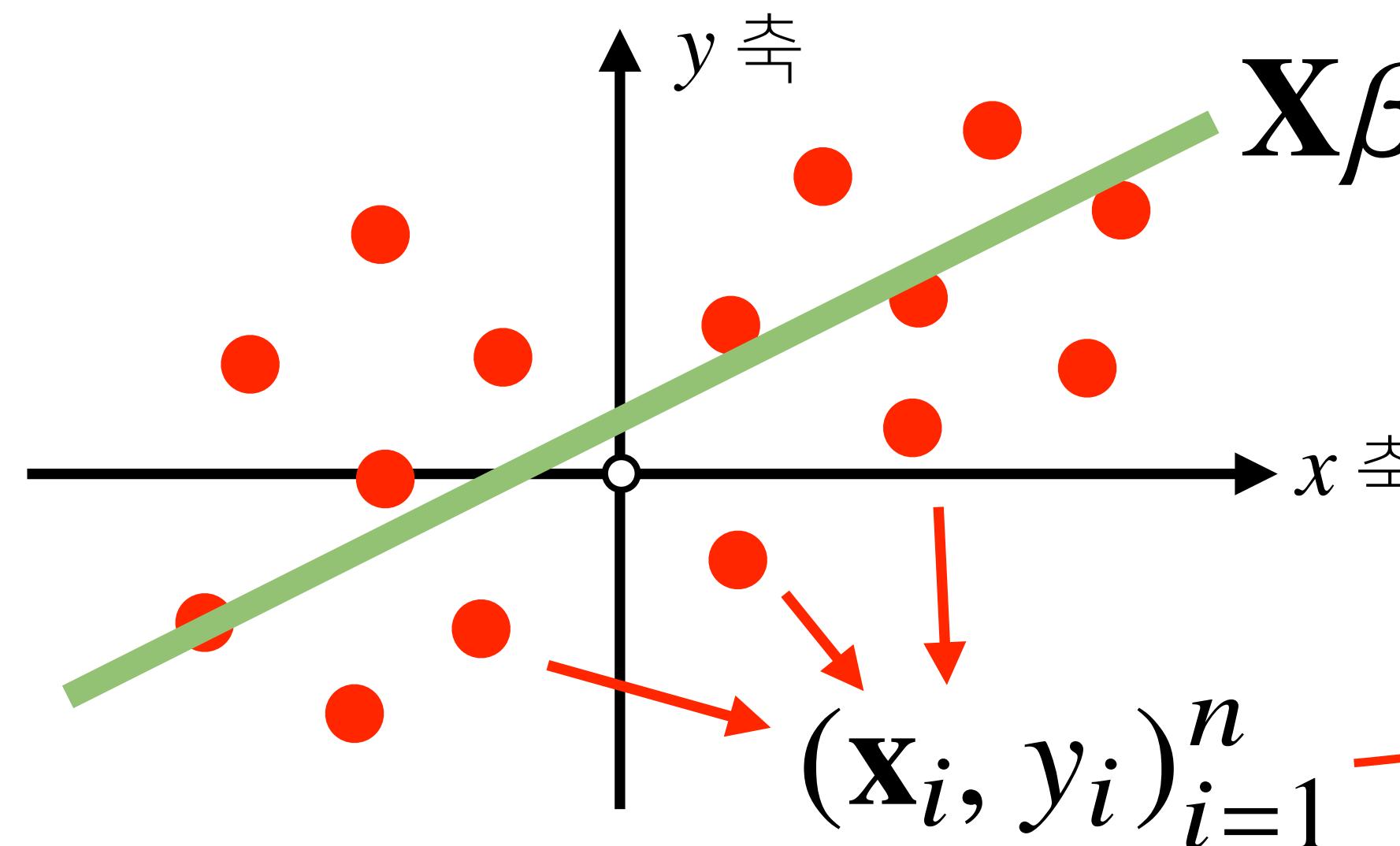
$$\Rightarrow \mathbf{x} = \mathbf{A}^+ \mathbf{b}$$

$$= \mathbf{A}^\top (\mathbf{A}\mathbf{A}^\top)^{-1} \mathbf{b}$$

×

응용2: 선형회귀분석

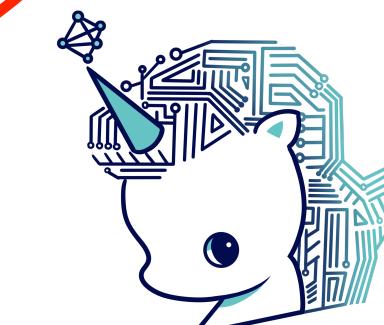
- `np.linalg.pinv` 를 이용하면 데이터를 선형모델(linear model)로 해석하는 선형회귀식을 찾을 수 있다



$n \geq m$ 인 경우: 데이터가 변수 개수보다 많거나 같아야 함

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

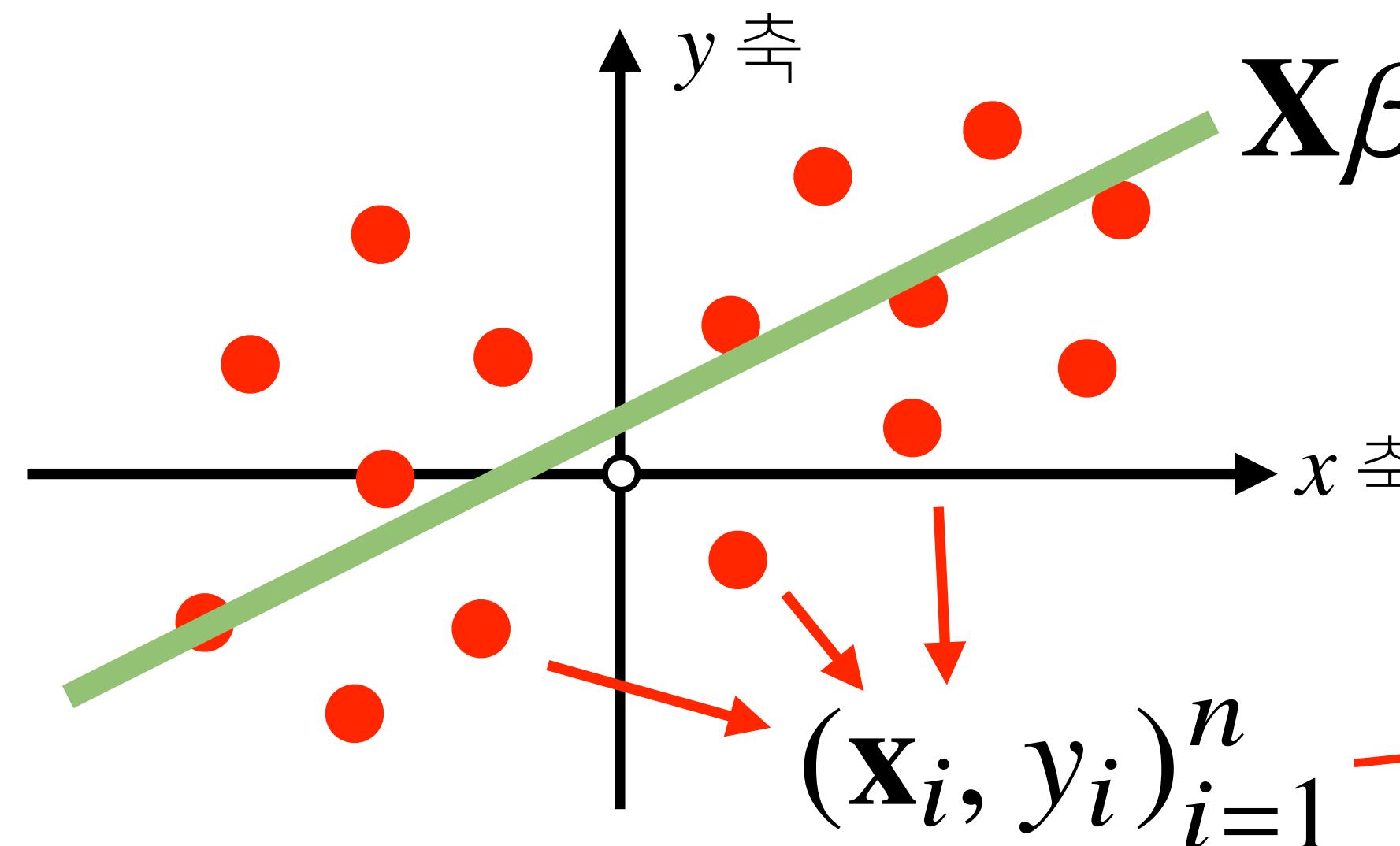
$\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$



선형회귀분석은 \mathbf{X} 와 \mathbf{y} 가 주어진 상황에서 계수 $\boldsymbol{\beta}$ 를 찾아야 한다

응용2: 선형회귀분석

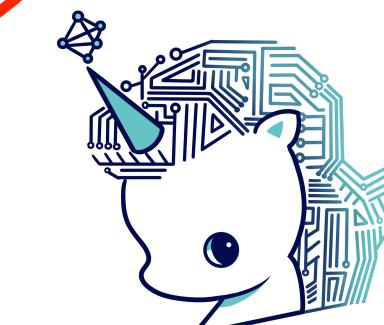
- `np.linalg.pinv` 를 이용하면 데이터를 선형모델(linear model)로 해석하는 선형회귀식을 찾을 수 있다



$n \geq m$ 인 경우: 데이터가 변수 개수보다 많거나 같아야 함

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \neq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

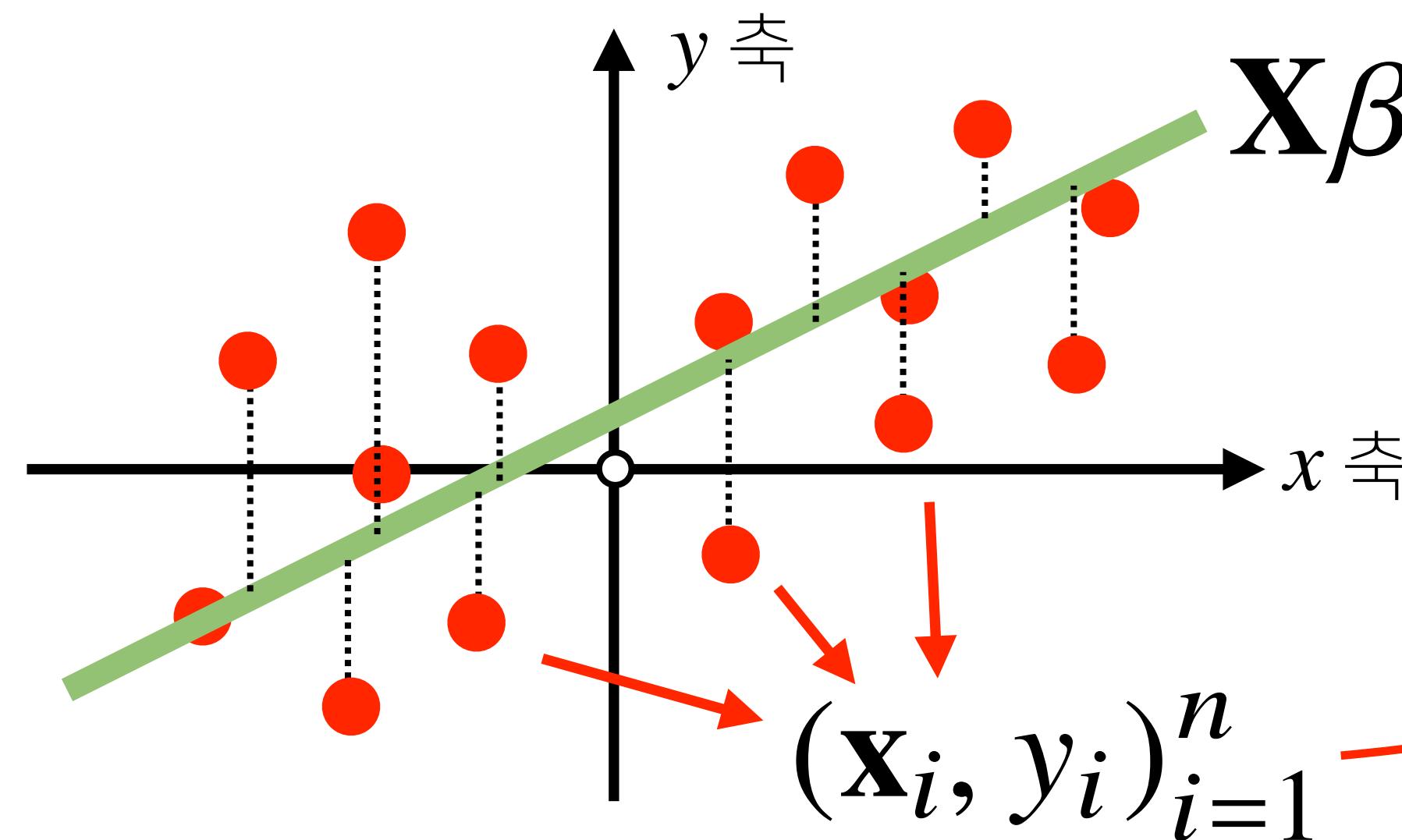
$$\mathbf{X}\boldsymbol{\beta} \neq \mathbf{y}$$



선형회귀분석은 연립방정식과 달리 행이 더 크므로 방정식을 푸는건 불가능

응용2: 선형회귀분석

- `np.linalg.pinv` 를 이용하면 데이터를 선형모델(linear model)로 해석하는 선형회귀식을 찾을 수 있다

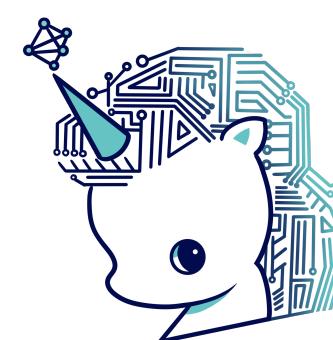


$n \geq m$ 인 경우: 데이터가 변수 개수보다 많거나 같아야 함

\times

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \neq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$
$$\mathbf{X}\beta = \hat{\mathbf{y}} \approx \mathbf{y}$$
$$\min_{\beta} \|\mathbf{y} - \hat{\mathbf{y}}\|_2$$
$$\Rightarrow \beta = \mathbf{X}^+ \mathbf{y}$$
$$= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

L_2 -노름을 최소화

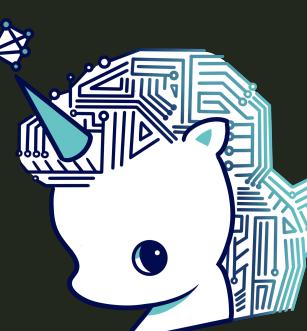


Moore-Penrose 역행렬을 이용하면
 \mathbf{y} 에 근접하는 $\hat{\mathbf{y}}$ 를 찾을 수 있다

응용2: 선형회귀분석

- `sklearn` 의 `LinearRegression` 과 같은 결과를 가져올 수 있다

```
1 # Scikit Learn 을 활용한 회귀분석  
2 from sklearn.linear_model import LinearRegression  
3 model = LinearRegression()  
4 model.fit(X, y)  
5 y_test = model.predict(x_test)  
6  
7 # Moore-Penrose 역행렬  
8  
9 beta = np.linalg.pinv(X) @ y  
10 y_test = np.append(x_test) @ beta
```

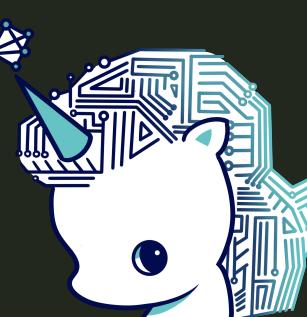


같은 방법이지만 결과가 다르게 된다. 왜일까?

×

응용2: 선형회귀분석

- `sklearn` 의 `LinearRegression` 과 같은 결과를 가져올 수 있다

```
1 # Scikit Learn 을 활용한 회귀분석
2 from sklearn.linear_model import LinearRegression
3 model = LinearRegression()
4 model.fit(X, y)
5 y_test = model.predict(x_test)  y절편(intercept) 항을 직접 추가해야 한다
6
7 # Moore-Penrose 역행렬
8 X_ = np.array([np.append(x, [1]) for x in X]) # intercept 항 추가
9 beta = np.linalg.pinv(X_) @ y
10 y_test = np.append(x, [1]) @ beta
```

THE END

다음 시간에 보아요!