

행렬

- 행렬(matrix): 벡터를 원소로 가지는 2차원 배열

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

```
X = np.array([[1,-2,3], [7,5,0], [-2,-1,-2]])  
# 행렬: 행벡터 n차원 -
```

- 특정 행(열) 고정하면 행(열)벡터라고 부름
- 전치행렬(transpose matrix): 행과 열의 인덱스가 바뀐 행렬 // 행벡터 => 열벡터 / 열벡터 => 행벡터
- 행렬은 여러 점들로 나타남
- 행렬의 행벡터 x_i 는 i 번째 데이터를 의미
- 벡터를 원소로 가지는 2차원 배열
- 같은 모양을 가지면 덧셈, 뺄셈 계산 가능 // 성분곱/스칼라곱도 벡터와 같다
- 행렬 곱셈(matrix multiplication)

i 번째 행벡터와 j 번째 열벡터 사이의 내적을 성분으로 가지는 행렬 계산

행렬곱은 각각 열의 개수와 행의 개수가 같아야 함

numpy에서는 @ 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]])
```

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

- 행렬 내적

넘파이의 np.inner는 i 번째 행벡터와 j 번째 행벡터 사이의 내적을 성분으로 가지는 행렬 계산

수학에서의 내적과 다름

```
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]])
```

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

- 행렬: 벡터공간에서 사용되는 연산자(operator)로 이해
- 행렬곱을 통해 벡터를 다른 차원의 공간 보낼 수 있다.
- 패턴을 추출하고 데이터를 압축할 수도 있다.
- 모든 선형변환(linear transform)은 행렬곱으로 계산할 수 있다.

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

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

$\mathbf{z} \qquad \qquad \mathbf{A} \qquad \qquad \mathbf{x}$

$\mathbb{R}^m : m\text{차원 공간} \qquad \mathbb{R}^n : n\text{차원 공간}$

- 역행렬

어떤 행렬 A 의 연산을 거꾸로 되돌리는 행렬

A^{-1} 이라고 표기 / `np.linalg.inv()` 이용

가우스 소거법(Fauss Elimination) / 가우스-조던 소거법(Fauss-Jordan Elimination) 이용해서 구함

행과 열 숫자가 같고 행렬식(determinant)이 0이 아닌 경우에만 계산

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \det(A) = ad - bc$$

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \det(A) = a \cdot \det \begin{pmatrix} e & f \\ h & i \end{pmatrix} - b \cdot \det \begin{pmatrix} d & f \\ g & i \end{pmatrix} + c \cdot \det \begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

역행렬 공식

이차정사각행렬 $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 에 대하여

$$ad - bc \neq 0 \text{ 이면 } A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

$ad - bc = 0$ 이면 행렬 A 의 역행렬은 없다.

○ 항등행렬: 주대각선의 원소가 모두 1이며 나머지 원소는 모두 0인 정사각 행렬

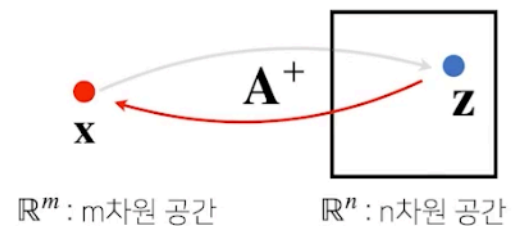
- 역행렬을 계산할 수 없다면 유사역행렬(pseudo-inverse) 또는 무어-펜로즈(Moore-Penrose) 역행렬 A^+ 이용

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

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



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



- $n > m$

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

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2$$

\vdots

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

$n \leq m$ 인 경우: 식이 변수 개수보다 작거나 같아야 함



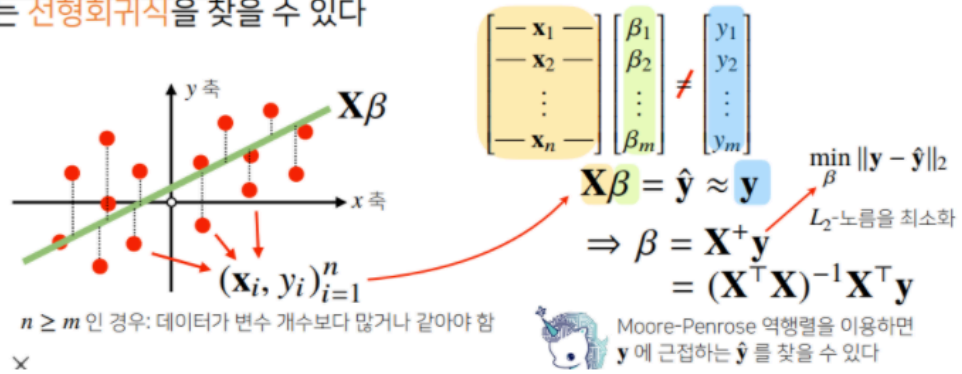
$n \leq m$ 이면 무어-펜로즈 역행렬을
 이용하면 해를 하나 구할 수 있다

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

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

- $n < m$

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



`sklearn.linear_model.LinearRegression`과 같은 결과 가져옴

`sklearn`에서는 y절편항을 직접 추가해야 함.

Scikit Learn을 활용한 회귀분석

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y)
y_test = model.predict(x_test)
```

Moore-Penrose 역행렬

```
X_ = np.array(np.append(x, [1]) for x in X) # intercept 항 직접 추가 필요
beta = np.linalg.pinv(X_) @ y
y_test = np.append(x, [1]) @ beta
```