2.1 데이터와 행렬

1. 벡터의 차원 != 배열의 차원
- 벡터의 차원 : 원소의 갯수
- 배열의 차원 : 원소의 갯수가 몇개이든, 한 줄로 나타낼 수 있으면 '1차원 배열'
, 가로와 세로가 있는 여러 줄의 직사각형 형태로 나타낼 수 있으면 '2차원 배열'
이라고 함

2. 예측문제의 입력데이터는 대부분 1줄의 열벡터로 변환
- 예측문제의 입력데이터는 대부분 벡터로 표시함
- 2차원 이미지데이터 -> 1차원 열벡터로 변환해 길게 늘어트린다.(reshape)

특징벡터

3. 행렬에선 열벡터를 행벡터로 바꿔 표시

특징행렬

2.2 벡터와 행렬의 연산

1. 행렬의 +,-

- 1) 같은 size끼리 +, - 는 Element-wise(요소별) 연산
- 2) 다른 사이즈, 특히 벡터/행렬 --- 스칼라 간 +,- => 브로드캐스팅
    - *벡터와 스칼라의 연산 : 관례적으로 스칼라를 벡터로 변환한 연산 허용 (c -> c*(1벡터))
- 3) element-wise로 평균을 빼준 벡터 = 평균제거벡터 (브로드캐스팅 연산)

2. 선형조합

3. 벡터와 벡터의 곱

- 1) 내적 (inner product, dot product) : element-wise product
- 2) 내적 결과는 스칼라
- 3) 내적은 가중합(weighted sum)을 구할 때 사용될 수 있음
    - 가중치의 합치 1일 경우, 내적 = 가중합 = 가중평균
- 4) 코사인유사도 ≒ 내적 = 두 벡터가 닮은 정도를 정량적으로 나타낸 값
- 5) 내적은 가중치 벡터와의 곱으로 선형회귀 계산에 활용됨

$$[x_1 \cdots x_n] \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

4. 특징행렬 (특징벡터를 전치해서 행렬구성) y_hat = Xw, X=특징행렬, w=가중치 벡터

- 특징벡터 : 각각의 꽃송이, 각각의 아파트 특징값은 열벡터로 구성. But, 특징행렬 구성 시, 특징벡터를 전치하여 행 방향으로 구성

5. 잔차

- ei = (yi - y_hati)
- 잔차의 크기 = RSS (잔차제곱합)

$RSS = e^T e = e_1^2 + \cdots + e_n^2$ = 잔차 크기

2.2 벡터와 행렬의 연산

6. Quadratic Form 이차형식 "행벡터 x 행렬 x 열벡터" square!

$x^T A x$ ⇐ Quadratic Form

↳ square!

$$x^T A x = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} a_{i,j} x_i x_j$$

= 가중치더해 총합

↳ 즉 $a_{ij} x_i x_j$

7. AB 의 이해

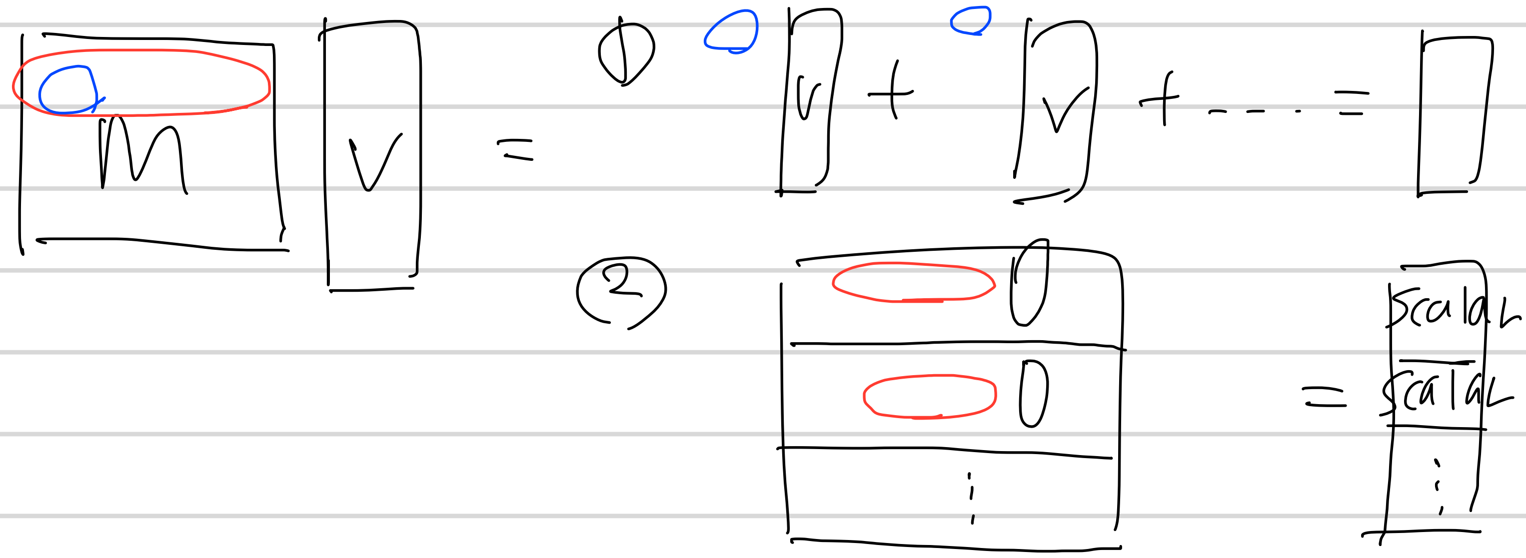
$A\ B$

① $\begin{bmatrix} -a_1^T- \\ -a_2^T- \end{bmatrix} B = \begin{bmatrix} -a_1^T B- \\ -a_2^T B- \end{bmatrix}$

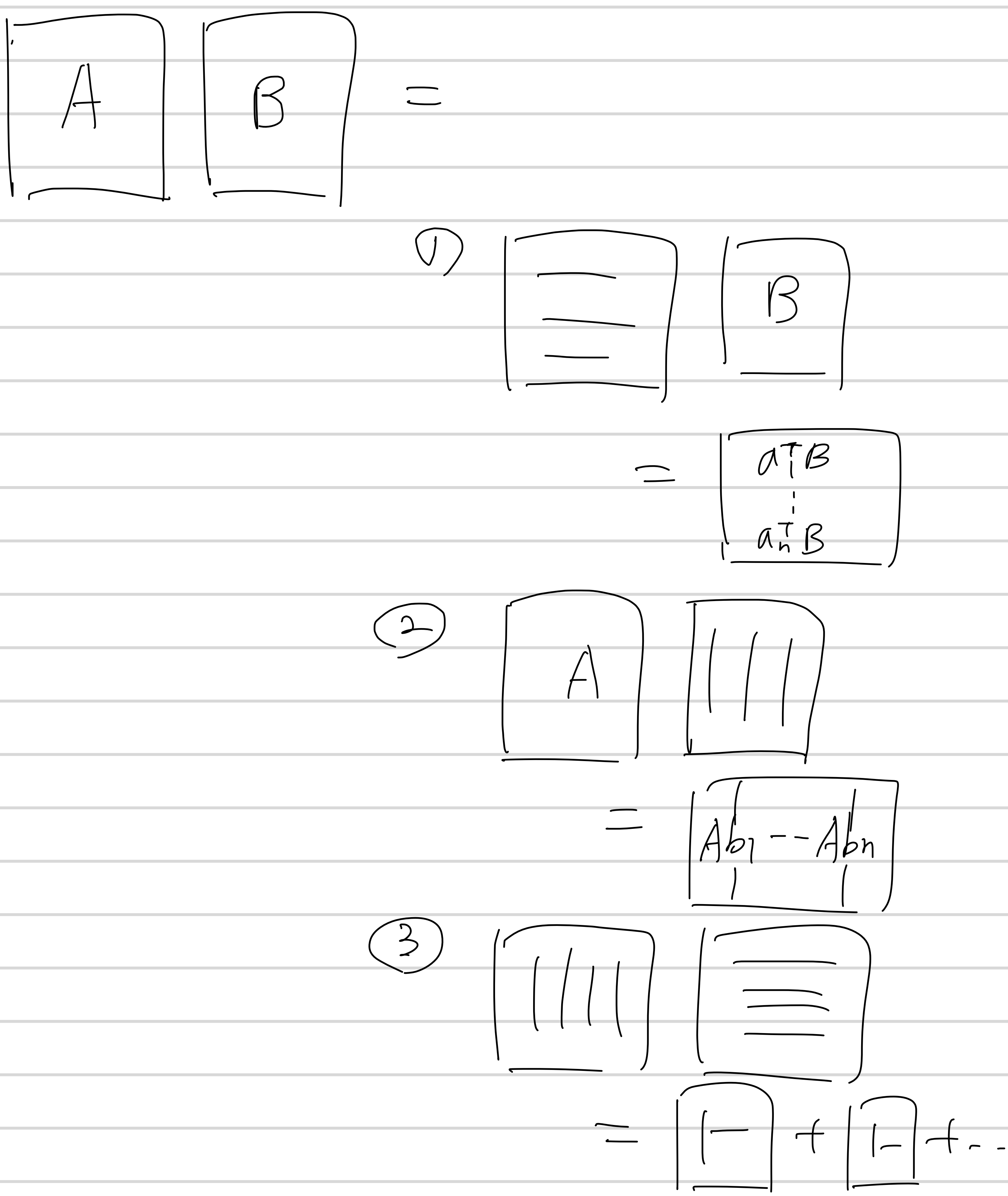
② $A \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} Ab_1 & Ab_2 \end{bmatrix}$

주로 이렇게 생각하자.

③ $\begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} -b_1^T- \\ -b_2^T- \end{bmatrix} =$



M
v
=
①
+
+ ---- =
②
scalar
= scalar



A
B
=
①
B
= a₁ᵀB
aₙᵀB
②
A
= Ab₁ -- Abₙ
③
= + + . .

2.3 행렬의 성질

* trace, determinant, norm

⇒ 행렬 크기를 나타내는 연산

1) 행렬의 부호(정부호)

(positive defi, positive semi defi)

- 정의
- 행렬 예시 2개(항등행렬, 대칭행렬)

2) 행렬의 크기

Q1. 왜 크기(길이, 사이즈)를 norm으로 하는가?

① 피타고라스 정리처럼 빗변의 길이 구하기?

② L1 norm은 왜 안되나?

- norm 정의 (||A||p)
- 벡터 크기 : L2 norm
- 행렬 크기 : frobeniums norm
    - 벡터의 (L2 norm)^2 = 내적값 = 벡터의 제곱합 (xT * x)

- code) numpy로 행렬 norm 구해보기

3) norm의 4가지 성질

Q2. 언제 등식성립?

- • 놈의 값은 0이상이다. 영행렬일 때만 놈의 값이 0이 된다.

$$\|A\| \geq 0 \quad (2.3.12)$$

- • 행렬에 스칼라를 곱하면 놈의 값도 그 스칼라의 절대값을 곱한 것과 같다.

$$\|\alpha A\| = |\alpha| \|A\| \quad (2.3.13)$$

- • 행렬의 합의 놈은 각 행렬의 놈의 합보다 작거나 같다.

$$\|A + B\| \leq \|A\| + \|B\| \quad (2.3.14)$$

- • 정방행렬의 곱의 놈은 각 정방행렬의 놈의 곱보다 작거나 같다.

$$\|AB\| \leq \|A\| \|B\| \quad (2.3.15)$$

4) Trace = 대각합

* $\|A\|^2 = tr(A^T A)$

내적의 trace

(연습문제 2.3.5)

- 정의
- 대각합은 정방행렬에서만 정의됨
- 항등행렬의 대각합
- 대각합 성질
    - tr(AB) = tr(BA)
    - tr(ABC) = tr(BCA) = tr(CAB) <= 'trace trick'
    - quadratic form의 trace

5) 행렬식

- 정의
- 성질
    - 전치행렬의 행렬식
    - 행렬 곱의 행렬식
    - 역행렬의 행렬식

2.4 선형 연립방
정식과
역행렬

2.4 선형 연립방정식과 역행렬

1) 선형연립방정식 = 행렬과 벡터의 곱

복수의 미지수를 포함하는 복수의 선형 방정식을 **선형 연립방정식(system of linear equations)** 또는 연립일차방정식이라고 한다.

다음은 3개의 미지수와 3개의 선형 방정식을 가지는 선형 연립방정식의 한 예다.

$$\begin{aligned} x_1 + x_2 \quad &= 2 \\ x_2 + x_3 &= 2 \\ x_1 + x_2 + x_3 &= 3 \end{aligned} \tag{2.4.1}$$

$x_1, x_2, \cdots, x_M$ 이라는 M 개의 미지수를 가지는 N 개의 선형 연립방정식은 일반적으로 다음과 같은 형태가 된다. 이 식에서 a와 b는 방정식의 계수다.

$$\begin{matrix} a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1M}x_M & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2M}x_M & = & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{N1}x_1 & + & a_{N2}x_2 & + \cdots + & a_{NM}x_M & = & b_N \end{matrix} \tag{2.4.2}$$

행렬과 벡터의 곱셈을 이용하면 위 선형 연립방정식은 다음처럼 간단하게 쓸 수 있다.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{2.4.3}$$

이 식에서

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{2.4.4}$$

라고 하면 다음처럼 쓸 수 있다.

$$Ax = b \tag{2.4.5}$$

A, x, b 는 각각 **계수행렬(coefficient matrix), 미지수벡터(unknown vector), 상수벡터(constant vector)**라고 부른다.

2) 역행렬

- 가역행렬 / 비가역행렬(특이행렬)

$(A^T)^{-1} = (A^{-1})^T$

- 대각행렬의 역행렬
- 역행렬의 성질
  - 전치행렬의 역행렬 ⟹ 대칭행렬의 역행렬도 대칭행렬 (?)

- code) 넘파이를 사용한 연립방정식 해결(역행렬 계산)
- code) lstsq방법 활용한 연립방정식 해결

3) 선형 연립방정식과 선형 예측모형

- 선형 예측모형의 가중치 벡터를 구하기 = 선형 연립방정식 풀기

  (Xw = y, w = (X)-1 y)

- code) 보스턴 집값 예측 문제 코딩

4) 미지수의 수와 방정식의 수

- 방정식의 수 = 미지수의 수
- 방정식의 수 < 미지수의 수
- 방정식의 수 > 미지수의 수 -> 최소자승법으로 근사값 구함

해를 구할수 없다면,

$\underset{m\times n}{A}\ \underset{n\times 1}{x^*} = b$

(m > n)

└> A는 square가 아니므로, 역행렬 X, ∴ x^* 못구할수도 ⟹ Pseudo inverse로 해결 (근사)

└> 최소제곱법! 접근임.
($y - \hat{y}$, $Ax^* - b$, e_i: 잔차를 최소화하는 x^*를 찾는 것)
e^Te의 최소화하는 x^*

잔차는 벡터이므로 최소자승문제에서는 벡터의 크기 중에서 **벡터의 놈(norm)을 최소화**하는 문제를 푼다. 앞 절에서 놈을 최소화하는 것은 놈의 제곱을 최소화하는 것과 같다고 했다. 여기에서는 잔차제곱합이 놈의 제곱이 된다.

$$e^Te = \|e\|^2 = (Ax-b)^T(Ax-b) \tag{2.4.40}$$

이 값을 최소화하는 x값은 수식으로 다음처럼 표현한다.

$$x = \arg\min_x e^Te = \arg\min_x (Ax-b)^T(Ax-b) \tag{2.4.41}$$

위 식에서 $\arg\min_x f(x)$는 함수 $f(x)$를 가장 작게 만드는 x값을 의미한다. 이러한 문제를 **최소자승문제(least square problem)**라고 한다.

2.4 선형 연립방정식과 역행렬

2.4 선형 연립방정식과 역행렬

5) 최소제곱법

- 방정식 갯수 > 미지수 갯수

-> 정학한 해 구할 수 없는 경우, 최소제곱법으로 가장 근접한 답을 찾는다.

- Ax = b의 해(x)가 없으니, Ax와 b가 가장 근접하게 하는 해(x)를 찾는다.

-> e = Ax - b 일 때, 잔차 e를 최소화 하는 해(x)를 찾는 것

- 잔차를 최소화 = 벡터의 최소화 = norm의 최소화

=> (norm)**2 최소화와 같다 = 내적의 최소화

- eT*e = ||e||^2 = (norm)**2 = (Ax-b)T (Ax-b)

근사해

- so, Ax와 b의 잔차의 최소값을 찾는다 -> 잔차의 최소값

= 잔차벡터 norm의 최소값 -> norm**2의 최소값 = 내적의 최소값

-> (Ax-b)T(Ax-b)의 최소값 찾기

- 이러한 문제를 최소제곱문제 라고 함.

- 최소제곱법 풀이 과정(Ax = b, x = ?)

위 식에서 $\arg\min_x f(x)$는 함수 $f(x)$를 가장 작게 만드는 x값을 의미한다. 이러한 문제를 **최소자승문제(least square problem)**라고 한다.

A^TA가 항상 정방 행렬이 된다는 점을 이용하여 다음과 같이 최소 자승 문제의 답이 어떤 형태가 되는지 살펴보자. 여기에서는 답의 형태만 살펴보고 엄밀한 증명은 하지 않을 것이다.

$$Ax \approx b \tag{2.4.42}$$

이 식의 양변에 A^T를 곱하면 각각 A^TAx와 A^Tb 가 된다. 이 두 개의 벡터의 값이 같다고 일단 가정하자.

$$A^TAx = A^Tb \tag{2.4.43}$$

만약 정방 행렬 A^TA의 역행렬 $(A^TA)^{-1}$ 이 존재한다면

$$(A^TA)^{-1}(A^TA)x = (A^TA)^{-1}A^Tb \tag{2.4.44}$$

이 식을 정리하면 다음과 같다.

$$x = ((A^TA)^{-1}A^T)b \tag{2.4.45}$$

매우 중요

위에서 보인 것은 수학적 증명이라고 할 수 없지만 엄밀한 수학적 증명을 통해 최소자승문제의 해를 구해도 위와 같은 결과를 얻을 수 있다. 자세한 내용은 행렬의 미분과 최적화를 공부한 뒤에 다루도록 한다.

여기에서 행렬 $(A^TA)^{-1}A^T$ 를 행렬 A의 **의사역행렬(pseudo inverse)**이라고 하며 다음처럼 A^+ 로 표기한다.

$$A^+ = (A^TA)^{-1}A^T \tag{2.4.46}$$

$$x = A^+b \tag{2.4.47}$$

넘파이의 `lstsq()` 명령은 사실 이러한 최소자승문제를 푸는 명령이다.

6) 최소제곱법의 계산 => 의사역행렬 (pseudo inverse) 활용

- 최소제곱법 접근을 통해, Ax^ = b 를 만족하는 근사해(x^)를 찾는 데,

이 때, A의 의사역행렬을 통해 x^를 구할 수 있다.

- x^ = (A+)b, A+ = 의사역행렬 이라 함

- code) 의사역행렬을 직접 계산해 최소제곱법 근사해 구하기

- code) np.linalg.lstsq 를 활용해 최소제곱법 근사해 구하기