**Single Variable Linear Regression-1**

|  |  |
| --- | --- |
| X(공부량) | Y(점수) |
| 2 | 40 |
| 4 | 60 |

위의 training data를 이용하여 학습을 하여 모델(regression)을 만든 후, test data의 입력값을 주면 결과를 예측한다.

**H(x) = wX + b (단변량 회귀분석)**

2\*w + b = 40

4\*w+ b = 60

H(X) = XW + b

X ∙ ab = Y

|  |
| --- |
| W |
| b |

|  |
| --- |
| 40 |
| 60 |

|  |
| --- |
| 2 1 |
| 4 1 |

=

2 X 2 행렬 ∙ 2 x 1 행렬 = 2 x 1 행렬

역행렬을 이용해서

|  |
| --- |
| w |
| b |

= 역행렬 ∙ Y

X ∙ ab = Y

(X)-1 ∙ X ∙ ab = (X)-1 ∙ Y

**ab = (X)-1 ∙ Y**

**Single Variable Linear Regression-1**

|  |  |
| --- | --- |
| X(독립변수) | Y(종속변수) |
| 32  64  96  118  126  144  152.5  158 | 18  24  61.5  49  52  105  130.3  125 |

**H(x) = wX + b (단변량 회귀분석)**

32\*w + b = 18

64\*w+ b = 24

…

H(X) = XW + b

X ∙ ab = Y

|  |
| --- |
| W |
| b |

|  |
| --- |
| 18  24  61.5  49  52  105  130.3  125 |

|  |
| --- |
| 32 1  64 1  96 1  118 1  126 1  144 1  152.5 1  158 1 |

=

8 X 2 행렬 ∙ 2 x 1 행렬 = 8 x 1 행렬

X ∙ ab = Y

(t(X)∙ X )-1 ∙ t(X) ∙ X ∙ ab = (t(X)∙ X )-1 ∙ t(X) ∙ Y

**ab = (t(X)∙ X )-1 ∙ t(X) ∙ Y**

**Multi Variable Linear Regression**

목표 ; 입력값이 다변량일 때의 Linear Regression

1. Multi-variable Linear Regression

Linear System : Set of Equations 연립방정식의 집합

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | X3 | Y(수명) |
| 60 kg | 5.5 ft | Yes(1) | 66 |
| 65 kg | 5.0 ft | No(0) | 74 |
| 55 kg | 6.0 ft | Yes(1) | 78 |
| 80 kg | 6.5 ft | No(0) | ? |

위의 training data를 이용하여 학습을 하여 모델(regression)을 만든 후, test data의 입력값을 주면 결과를 예측한다.

**H(x) = wX + b (단변량 회귀분석)**

**H(x1, x2, x3) = w1 x1 + w2 x2 + w3 x3 + b (다변량 회귀분석)**

60\*w1 + 5.5\*w2 +1\*w3=66

65\*w1 + 5.0\*w2 +0\*w3=74

55\*w1 + 6.0\*w2 +1\*w3=78

* Matrix 를 이용 (선형대수)

|  |
| --- |
| w1 |
| w2 |
| w3 |

( x1 x2 x3 ) ∙ = (x1 w1 + x2 w2 + x3 w3 )

H(X) = XW

|  |
| --- |
| w1 |
| w2 |
| w3 |

|  |
| --- |
| 66 |
| 74 |
| 78 |

|  |
| --- |
| 60 5.5 1 |
| 65 5.0 0 |
| 55 6.0 1 |

=

4 X 3 행렬 ∙ 3 x 1 행렬 = 4 x 1 행렬

역행렬을 이용해서

|  |
| --- |
| w1 |
| w2 |
| w3 |

= 역행렬 ∙ Y

1. 소스

import numpy as np

# column vector

c = np.array([1,2,3])

print(c.shape)

#obsaining a particular entry

print(c[0])

#row vector

r = np.array([[1,2,3]])

r

print(r.shape)

print(r[0,1], r[0][1])

#creating a atrix with all zeros

a = np.zeros((2,2))

print(a)

# creating a matrix with all ones

b = np.ones((2,2))

print(b)

#creating a matrix filled with the same constant

c = np.full((2,2),7)

print(c)

# creating a matrix with random values

d = np.random.random((2,2))

print(d)

#creating a matrix

A = np.array([[1,2],[3,4],[5,6]])

A

#creating another matrix

B = np.array([[11,12,13,14],[15,16,17,18]])

B

# trainspose a matrix

A.T

# matrix-matrix multiplication

A.dot(B)

A@B # A\*B(요수들간의 곱)와 다른 행렬의 곱

B @ A

#coefficient matrix A and a vector b

x1 = np.array([60,65,55])

x2 = np.array([5.5,5.0, 6.0])

x3 = np.array([1,0,1])

A = np.c\_[x1, x2, x3]

b = np.array([66,70,78])

print(A)

print(b)

#computing an inverse

from numpy.linalg import inv

A\_inv = inv(A)

A\_inv

# wrong matrix multiplication (요소간 곱. 행렬의 곱은 dot이나 @)

A\_inv \* A

A\_inv @ A

A\_inv.dot(A)

A @ A\_inv

A.dot(A\_inv)

x = A\_inv @ b

x # 연립방정식의 가중치값 나옴

# a better way to solve the same linear system

from numpy.linalg import solve

x = solve(A, b) #함수를 이용해서 나옴

x