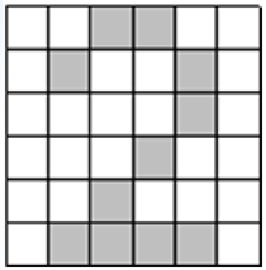
머신러닝/딥러닝을 위한

파이썬(Python)

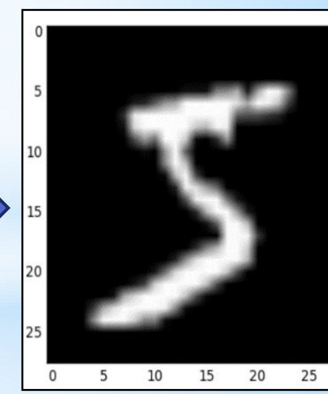
– numpy • matplotlib –

행렬(matrix) 필요성



0	0	1	1	0	0
0	1	0	0	1	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	1	1	1	0

5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,166,255,247,127,0,0
,0,0,0,0,0,0,0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,0,0,0,0,0,0,0,0,0,0,0,49,238,253,253
,253,253,253,253,253,253,251,93,82,82,56,39,0,0,0,0,0,0,0,0,0,0,0,0,18,219,253,253,253,253,253,198,182,247,241,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205,11,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,1,154,253,9
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,139,253,190,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,
190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,81,240,253,253,119,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,45,186,253,253,150,27,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,93,252,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,249,253,249,64,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,148,229,253,2
53,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,201,78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,171,219,253,253,253,253,195,80,9,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,55,172,226,253,253,253,253,244,133,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,253,253,253,212,135
,132,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0



라이브러리 – overview

- 파이썬 코드에서 라이브러리 사용법

```
import numpy

A = numpy.array([1,2])

print("A ==", A, ", type ==", type(A))
```

```
A == [1 2] , type == <class 'numpy.ndarray'>
```

```
import numpy as np

A = np.array([1,2])

print("A ==", A, ", type ==", type(A))
```

```
A == [1 2] , type == <class 'numpy.ndarray'>
```

```
from numpy import exp

result = exp(1)

print("result ==", result, ", type ==", type(result))
```

```
result == 2.7182818284590452 , type == <class 'numpy.float64'>
```

```
from numpy import *

result = exp(1) + log(1.7) + sqrt(2)

print("result ==", result, ", type ==", type(result))
```

```
result == 4.6631236418943111 , type == <class 'numpy.float64'>
```

- numpy
  - vector / matrix 생성
  - 행렬 곱 (dot product)
  - broadcast
  - index / slice / iterator
  - useful function (loadtxt(), rand(), argmax(), …)

라이브러리 – numpy

- numpy

  => numpy는 머신러닝 코드 개발 할 경우 자주 사용되는 벡터, 행렬 등을 표현하고 연산할 때 반드시 필요한 라이브러리

- numpy vs list

  => 머신러닝에서 숫자, 사람, 동물 등의 인식을 하기 위해서는 이미지(image)데이터를 행렬(matrix)로 변환하는 것이 중요함

  => 행렬(matrix)을 나타내기 위해서는 리스트(list)를 사용할 수도 있지만, 행렬 연산이 직관적이지 않고 오류 가능성이 높기 때문에, 행렬 연산을 위해서는 numpy 사용이 필수임

행렬 연산

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad A + B = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

```
import numpy as np

# 리스트로 행렬 표현

A = [ [1, 0], [0, 1] ]
B = [ [1, 1], [1, 1] ]

A + B    # 행렬 연산이 아닌 리스트 연산
```

```
[[1, 0], [0, 1], [1, 1], [1, 1]]
```

```
# numpy matrix, 직관적임

A = np.array([ [1, 0], [0, 1] ])
B = np.array([ [1, 1], [1, 1] ])

A + B    # 행렬 연산
```

```
array([[2, 1],
       [1, 2]])
```

라이브러리 – numpy vector (1차원 배열)

- 벡터(vector) 생성

  => vector는 np.array([…])를 사용하여 생성함 (import numpy as np)

  => 머신러닝 코드 구현 시, 연산을 위해서 vector, matrix 등의 형상(shape), 차원(dimension)을 확인하는 것이 필요함

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

# vector A, B 출력
print("A ==", A, ", B == ", B)

# vector A, B 형상 출력 => shape
print("A.shape ==", A.shape, ", B.shape ==", B.shape)

# vector A, B 차원 출력 => ndim
print("A.ndim ==", A.ndim, ", B.ndim ==", B.ndim)
```

```
A == [1 2 3] , B ==  [4 5 6]
A.shape == (3,) , B.shape == (3,)
A.ndim == 1 , B.ndim == 1
```

- 벡터(vector) 산술연산

  => vector 간 산술연산(+, –, X, /)은 벡터의 각각의 원소에 대해서 행해짐

```
# vector 산술 연산

print("A + B ==", A+B)
print("A - B ==", A-B)
print("A * B ==", A*B)
print("A / B ==", A+B)
```

```
A + B == [5 7 9]
A - B == [-3 -3 -3]
A * B == [ 4 10 18]
A / B == [5 7 9]
```

라이브러리 – numpy matrix (행렬)

- 행렬(matrix) 생성

  => matrix는 vector와 마찬가지로 np.array([[…], […], …])를 사용하여 생성함 (import numpy as np)

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad B = \begin{pmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \end{pmatrix}$$

shape 2 X 3 shape 2 X 3

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

# matrix A, B 형상 출력 => shape
print("A.shape ==", A.shape, ", B.shape ==", B.shape)

# matrix A, B 차원 출력 => ndim
print("A.ndim ==", A.ndim, ", B.ndim ==", B.ndim)
```

```
A.shape == (2, 3) , B.shape == (2, 3)
A.ndim == 2 , B.ndim == 2
```

- 형 변환 (reshape)

  => vector를 matrix로 변경하거나 matrix를 다른 형상의 matrix로 변경하기 위해서는 reshape() 사용하여 행렬의 shape 을 변경하여야 함

$$D = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad D.reshape(3,2) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

shape 2 X 3 shape 3 X 2

```
# vector 생성

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

# vector C형상 출력 => shape
print("C.shape ==", C.shape)

# vector를 (1,3) 행렬로 형 변환
C = C.reshape(1, 3)

print("C.shape ==", C.shape)
```

```
C.shape == (3,)
C.shape == (1, 3)
```

라이브러리 – numpy matrix (행렬)

- 형 변환 reshape

=> 파이썬 넘파이에서는 numpy.reshape(-1, …) 형식을 이용하여 … 으로 주어지는 열(column) 을 가지는 행렬로 형 변환을 시킬 수 있음

```
import numpy as np

A = np.array([ [ 1, 2, 3, 4 ],
               [ 10, 20, 30, 40 ],
               [ 100, 200, 300, 400] ])

print(A.shape)
```

```
(3, 4)
```

```
B = A.reshape(-1, 3)

print(B.shape)
print(B)
```

```
(4, 3)
[[  1   2   3]
 [  4  10  20]
 [ 30  40 100]
 [200 300 400]]
```

```
C = A.reshape(-1, 6)

print(C.shape)
print(C)
```

```
(2, 6)
[[  1   2   3   4  10  20]
 [ 30  40 100 200 300 400]]
```

```
D = A.reshape(-1, 5)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-5ced01d6a714> in <module>
----> 1 D = A.reshape(-1, 5)

ValueError: cannot reshape array of size 12 into shape (5)
```

라이브러리 – numpy broadcast

- 행렬의 사칙연산은 기본적으로 두 개의 행렬 크기가 같은 경우에만 수행할 수 있음. 그러나 numpy 에서는 크기가 다른 두 행렬간에도 사칙연산(+, −, *, /)을 할 수 있는데 이를 브로드캐스트(broadcast)라고 지칭함

  => 차원이 작은 쪽이 큰 쪽의 행 단위로 반복적으로 크기를 맞춘 후에 계산

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $b = 5$ 인 경우,

$$A + B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + 5 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ 8 & 9 \end{pmatrix}$$

broadcast

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

print(A+b)
```

```
[[6 7]
 [8 9]]
```

$C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $D = (4 \quad 5)$ 인 경우,

$$C + D = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + (4 \quad 5) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 4 & 5 \\ 4 & 5 \end{pmatrix} = \begin{pmatrix} 5 & 7 \\ 7 & 9 \end{pmatrix}$$

broadcast

```
C = np.array([ [1, 2], [3, 4] ])
D = np.array([4, 5])

print(C+D)
```

```
[[5 7]
 [7 9]]
```

[예제 1_1] 벡터 A, B, T 를 이용하여 다음을 계산하고 넘파이를 이용하여 결과를 확인하시오

$$A = (1 \quad 2)$$
$$B = (3 \quad 4)$$
$$T = (5 \quad 6)$$

$(1-A)*A$ $\qquad$ $T*(1-A)*A$

[예제 1_2] 행렬 C, D, T 를 이용하여 다음을 계산하고 넘파이를 이용하여 결과를 확인하시오

$$C = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$
$$D = \begin{pmatrix} 4 \\ 5 \end{pmatrix}$$
$$T = \begin{pmatrix} 7 \\ 8 \end{pmatrix}$$

$(1-C)*C$ $\qquad$ $T*(1-C)*C$

라이브러리 – numpy 행렬 곱(dot product)

- 행렬 곱(dot product)

=> A 행렬과 B 행렬의 행렬 곱 (dot product)는 np.dot(A,B) 나타내며, 행렬 A의 열 벡터와 B 행렬의 행 벡터가 같아야 함. 만약 같지 않다면 reshape 또는 전치행렬(transpose)등을 사용하여 형 변환을 한 후에 행렬 곱 실행해야 함

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad B = \begin{pmatrix} -1 & -2 \\ -3 & -4 \\ -5 & -6 \end{pmatrix}$$

형상 2 X 3 3 X 2

$$A \cdot B = \begin{pmatrix} -22 & -28 \\ -49 & -64 \end{pmatrix}$$

(2 X 3) • (3 X 2) = (2 X 2)

```
A = np.array([ [1, 2, 3], [4, 5, 6] ])  # 2X3 행렬
B = np.array([ [-1, -2], [-3, -4], [-5, -6] ])  # 3X2 행렬

# (2X3) dot porduct (3X2) == (2X2) 행렬
C = np.dot(A, B)  # 행렬 곱 수행

# matrix A, B 형상 출력 => shape
print("A.shape ==", A.shape, ", B.shape ==", B.shape)
print("C.shape ==", C.shape)
print(C)
```

```
A.shape == (2, 3) , B.shape == (3, 2)
C.shape == (2, 2)
[[-22 -28]
 [-49 -64]]
```

[예제 2_1] 행렬 X, W, T 그리고 scalar b 에 대하여 Y, E 를 계산하고 넘파이를 이용하여 결과를 확인하시오

(Error 가 발생하는 경우, Error 원인이 무엇인지 파악하시오)

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad W = \begin{pmatrix} 0 & 1 & 1 \\ 3 & 2 & 1 \end{pmatrix}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}, \quad b = -1$$

$$Y = X \bullet W + b$$

$$E = T - Y$$

[예제 2_2] 다음 error 원인은 무엇인지 파악하시오

```
import numpy as np

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

B = np.array([10, 20, 30])

C = A.T

D = np.dot(C, B)
```

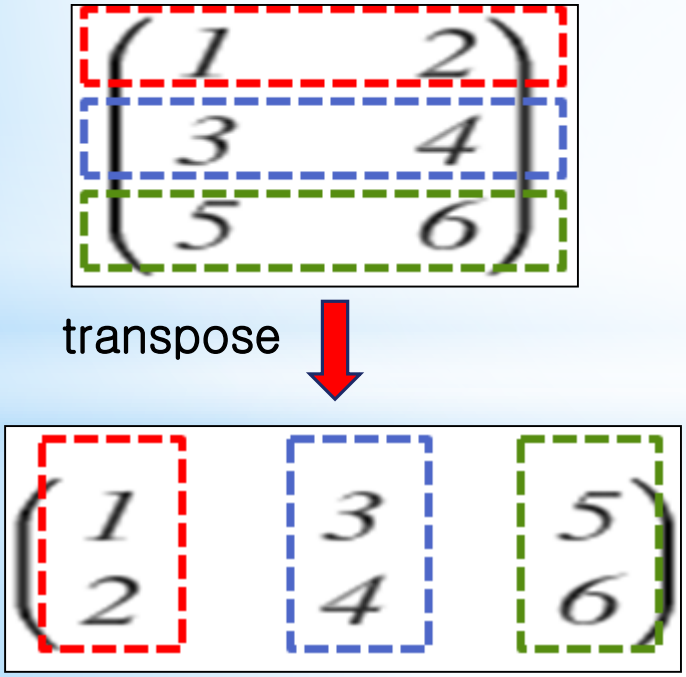
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-4-564ded330d43> in <module>
      5 C = A.T
      6
----> 7 D = np.dot(C, B)

ValueError: shapes (3,2) and (3,) not aligned: 2 (dim 1) != 3 (dim 0)
```

라이브러리 – numpy 전치행렬 (transpose)

- 전치행렬 (transpose)

  => 어떤 행렬의 전치행렬(transposed matrix)은 원본 행렬의 열은 행으로, 행은 열로 바꾼 것으로서, 원본 행렬을 A라고 하면 전치행렬은 A^T 로 나타냄.

  즉, 1행은 1열로 바꾸고 2행은 2열로, 3행은 3열로 바꾼 행렬을 의미




```
A = np.array([ [1, 2], [3, 4], [5, 6] ])  # 3X2행렬

B = A.T    # A의 전치행렬, 2X3 행렬

print("A.shape ==", A.shape, ", B.shape ==", B.shape)
print(A)
print(B)
```

```
A.shape == (3, 2) , B.shape == (2, 3)
[[1 2]
 [3 4]
 [5 6]]
[[1 3 5]
 [2 4 6]]
```

```
# vector 전치행렬

C = np.array([1, 2, 3, 4, 5])  # vector, matrix 아님
D = C.T      # C 는 vector 이므로 transpose 않됨

E = C.reshape(1, 5)    #  1X5 matrix
F = E.T    # E 의 전치행렬

print("C.shape ==", C.shape, ", D.shape ==", D.shape)
print("E.shape ==", E.shape, ", F.shape ==", F.shape)
print(F)
```

```
C.shape == (5,) , D.shape == (5,)
E.shape == (1, 5) , F.shape == (5, 1)
[[1]
 [2]
 [3]
 [4]
 [5]]
```

라이브러리 – numpy 행렬 indexing / slicing

- 행렬 원소 접근 (I)

  => 행렬 원소를 명시적(explicit)으로 접근하기 위해서는 *리스트(list)*에서처럼, 인덱싱 / 슬라이싱 모두 사용가능 함.

  [예제 1] A[0, 0] 은 1행 1열,
  A[2, 1] 은 3행 2열임

  [예제 2] A[: , 0] 은 모든 행 1열을 나타냄

  [예제 3] A[0:-1, 1:2] 인덱스 0인 1 행부터, 인덱스 -1-1=-2인 2행까지의 모든 데이터, 그리고 인덱스 1인 2열부터 인덱스 2-1=1 인 2 열까지의 모든 데이터

- [예제 4] A[:, :] 은 모든 행, 모든 열

```
A = np.array([10, 20, 30, 40, 50, 60]).reshape(3, 2)

print("A.shape ==", A.shape)
print(A)
```

```
A.shape == (3, 2)
[[10 20]
 [30 40]
 [50 60]]
```

```
print("A[0, 0] ==", A[0, 0] )
print("A[2, 1] ==", A[2, 1] )
```

```
A[0, 0] == 10
A[2, 1] == 60
```

```
print("A[0:-1, 1:2] ==", A[0:-1, 1:2])
```

```
A[0:-1, 1:2] == [[20]
 [40]]
```

```
print("A[ :, 0] ==", A[:, 0])
print("A[ :, :] ==", A[:, :])
```

```
A[ :, 0] == [10 30 50]
A[ :, :] == [[10 20]
 [30 40]
 [50 60]]
```

라이브러리 – numpy 행렬 iterator

- 행렬 원소 접근 (II)

  => 명시적(explicit) 인덱싱 / 슬라이싱 이외에, 행렬 모든 원소를 access 하는 경우에는 iterator 사용가능

  => numpy iterator는 C++, Java iterator 처럼 next() 메서드를 통해 데이터 값을 처음부터 끝까지 순차적으로 읽어 들이는 방법을 제공

  2X4 행렬인 경우, 인덱스 변화 순서

  (0,0) => (0,1) => (0,2) => (0,3)

  => (1,0) => (1,1) => (1,2) => (1,3)

  순서로 access

```
import numpy as np

A = np.array([ [10, 20, 30, 40], [50, 60, 70, 80] ])

print(A, '₩n')

# 행렬 A 의 iterator 생성

it = np.nditer(A, flags=['multi_index'], op_flags=['readwrite'])

while not it.finished:

    idx = it.multi_index

    print('index = ', idx, ' , type(idx) = ' , type(idx), ' , value = ', A[idx])

    it.iternext()
```

```
[[10 20 30 40]
 [50 60 70 80]]

index =  (0, 0)  , type(idx) =  <class 'tuple'>  , value =  10
index =  (0, 1)  , type(idx) =  <class 'tuple'>  , value =  20
index =  (0, 2)  , type(idx) =  <class 'tuple'>  , value =  30
index =  (0, 3)  , type(idx) =  <class 'tuple'>  , value =  40
index =  (1, 0)  , type(idx) =  <class 'tuple'>  , value =  50
index =  (1, 1)  , type(idx) =  <class 'tuple'>  , value =  60
index =  (1, 2)  , type(idx) =  <class 'tuple'>  , value =  70
index =  (1, 3)  , type(idx) =  <class 'tuple'>  , value =  80
```

라이브러리 – numpy useful function (I)

- seperator 로 구분된 파일에서 데이터를 읽기 위한 numpy.loadtxt(…)

  [예제] 다음과 같이 콤마(,)로 분리된 데이터 파일을 read 하기위해서는 np.loadtxt("파일이름",delimiter=',') 호출함. 리턴값은 행렬이기 때문에 인덱싱 또는 슬라이싱을 이용하여 데이터를 분리 할 필요 있음

  => 머신러닝 코드에서 입력데이터와 정답데이터를 분리하는 프로그래밍 기법

```
loaded_data = np.loadtxt('./data-01.csv', delimiter=',', dtype=np.float32)

x_data = loaded_data[ :, 0:-1]
t_data = loaded_data[ :, [-1]]

# 데이터 차원 및 shape 확인
print("x_data.ndim = ", x_data.ndim, ", x_data.shape = ", x_data.shape)
print("t_data.ndim = ", t_data.ndim, ", t_data.shape = ", t_data.shape)
```

```
x_data.ndim =  2 , x_data.shape =  (25, 3)
t_data.ndim =  2 , t_data.shape =  (25, 1)
```

./data-01.csv

```
73,80,75,152
93,88,93,185
89,91,90,180
96,98,91,196
73,66,70,142
53,46,55,101
69,74,77,149
47,56,60,115
87,79,90,175
79,70,88,164
69,70,73,141
70,65,74,141
93,95,91,184
79,80,73,152
70,73,78,148
93,89,96,192
78,75,68,147
81,90,93,183
88,92,86,177
78,83,77,159
82,86,90,177
86,82,89,175
78,83,85,175
76,83,71,149
96,93,95,192
```

라이브러리 – numpy useful function (II)

np.random.rand(…)

```
# 0 ~ 1 사이의 random number 발생

random_number1 = np.random.rand(3)
random_number2 = np.random.rand(1, 3)
random_number3 = np.random.rand(3, 1)

print("random_number1 ==", random_number1, ", random_number1.shape ==", random_number1.shape)
print("random_number2 ==", random_number2, ", random_number2.shape ==", random_number2.shape)
print("random_number3 ==", random_number3, ", random_number3.shape ==", random_number3.shape)
```

```
random_number1 == [0.75578495 0.91904082 0.4831561 ] , random_number1.shape == (3,)
random_number2 == [[0.9630389  0.80881622 0.71721651]] , random_number2.shape == (1, 3)
random_number3 == [[0.77989207]
 [0.58428724]
 [0.22968443]] , random_number3.shape == (3, 1)
```

np.sum(…), np.exp(…), np.log(…)

```
X = np.array([2, 4, 6, 8])

print("np.sum(X) ==", np.sum(X))
print("np.exp(X) ==", np.exp(X))
print("np.log(X) ==", np.log(X))
```

```
np.sum(X) == 20
np.exp(X) == [   7.3890561    54.59815003  403.42879349 2980.95798704]
np.log(X) == [0.69314718 1.38629436 1.79175947 2.07944154]
```

라이브러리 – numpy useful function (III)

np.max(…), np.min(…), np.argmax(…), np.argmin(…)

```
X = np.array([2, 4, 6, 8])

print("np.max(X) ==", np.max(X))
print("np.min(X) ==", np.min(X))
print("np.argmax(X) ==", np.argmax(X))
print("np.argmin(X) ==", np.argmin(X))
```

```
np.max(X) == 8
np.min(X) == 2
np.argmax(X) == 3
np.argmin(X) == 0
```

np.ones(…), np.zeros(…)

```
A = np.ones([3, 3])

print("A.shape ==", A.shape, ", A ==", A)

B = np.zeros([3, 2])

print("B.shape ==", B.shape, ", B ==", B)
```

```
A.shape == (3, 3) , A == [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
B.shape == (3, 2) , B == [[0. 0.]
 [0. 0.]
 [0. 0.]]
```

```
X = np.array([ [2, 4, 6], [1, 2, 3], [0, 5, 8] ])

print("np.max(X) ==", np.max(X, axis=0)) # axis=0, 열기준
print("np.min(X) ==", np.min(X, axis=0)) # axis=0, 열기준

print("np.max(X) ==", np.max(X, axis=1)) # axis=1, 행기준
print("np.min(X) ==", np.min(X, axis=1)) # axis=1, 행기준

print("np.argmax(X) ==", np.argmax(X, axis=0)) # axis=0, 열기준
print("np.argmin(X) ==", np.argmin(X, axis=0)) # axis=0, 열기준

print("np.argmax(X) ==", np.argmax(X, axis=1)) # axis=1, 행기준
print("np.argmin(X) ==", np.argmin(X, axis=1)) # axis=1, 행기준
```

```
np.max(X) == [2 5 8]
np.min(X) == [0 2 3]
np.max(X) == [6 3 8]
np.min(X) == [2 1 0]
np.argmax(X) == [0 2 2]
np.argmin(X) == [2 1 1]
np.argmax(X) == [2 2 2]
np.argmin(X) == [0 0 0]
```

$$X = \begin{pmatrix} 2 & 4 & 6 \\ 1 & 2 & 3 \\ 0 & 5 & 8 \end{pmatrix}$$

[예제 3] X 는 다음과 같을때 (1)~(6)을 계산하고 넘파이를 이용하여 결과를 확인하시오

(1) A = X.reshape(-1, 3)

(2) B = A[0:2 , 1:2]

(3) C = A[1: , :-1]

(4) D = np.argmax(A, 1)

(5) E = C.reshape(-1,6)

(6) F = np.sum(E)

$$X = \begin{pmatrix} 7 & 5 & 3 & 1 \\ 1 & 2 & 3 & 4 \\ 4 & 0 & 2 & 8 \end{pmatrix}$$

라이브러리 – matplotlib, scatter plot

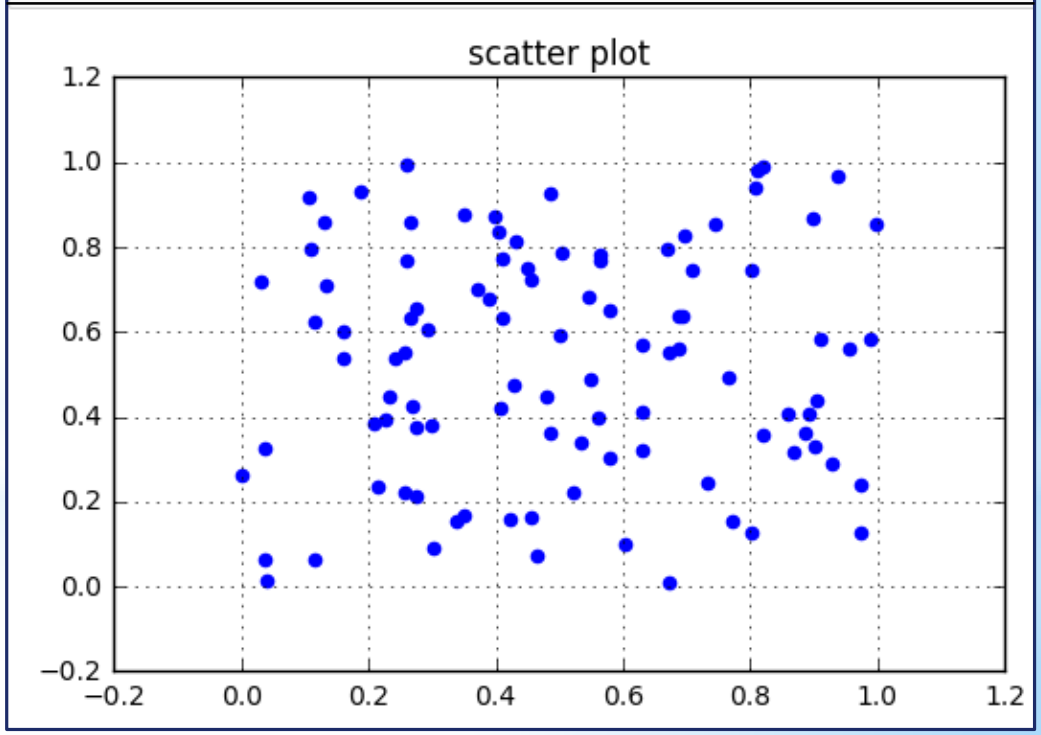
- 실무에서는 머신러닝 코드를 구현하기 전에,

  => 입력 데이터의 분포와 모양을 먼저 그래프로 그려보고, 데이터의 특성과 분포를 파악 한 후 어떤 알고리즘을 적용할 지 결정하고 있음

  => 데이터 시각화를 위해서는 matplotlib 라이브러리를 사용함

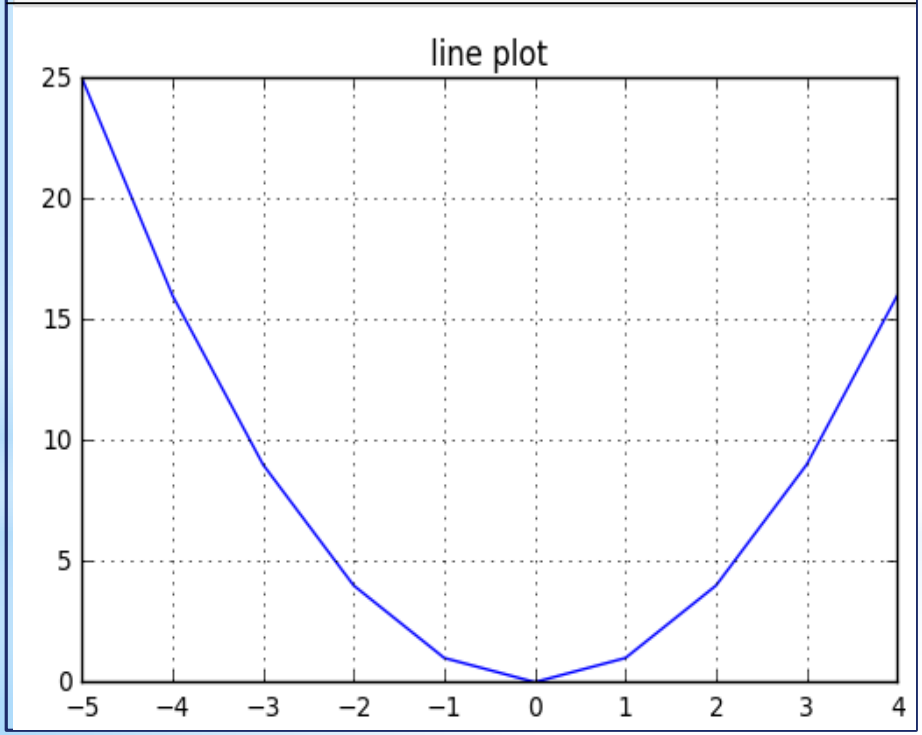
  => 일반적으로 line plot, scatter plot 등을 통해 데이터의 분포와 형태를 파악함

```
import matplotlib.pyplot as plt
import numpy as np

# 주피터 노트북을 사용하는 경우 노트북 내부에 그림 표시
%matplotlib inline

# x data,  y data 생성
x_data = np.random.rand(100)
y_data = np.random.rand(100)

plt.title('scatter plot')
plt.grid()
plt.scatter(x_data, y_data, color='b', marker='o')
plt.show()
```



라이브러리 – matplotlib, line plot

```
import matplotlib.pyplot as plt

# 주피터 노트북을 사용하는 경우 노트북 내부에 그림 표시
%matplotlib inline

x_data = [ x for x in range(-5,5) ]
y_data = [ y*y for y in range(-5,5) ]

plt.title('line plot')
plt.grid()
plt.plot(x_data, y_data, color='b')
plt.show()
```




```
import matplotlib.pyplot as plt


# 주피터 노트북을 사용하는 경우 노트북 내부에 그림 표시
%matplotlib inline

x_data = [ -3,  -2, -1, 0, 1, 2,   3,  4, 5, 6, 7,  8, 9 ]
y_data = [ -8, -13, -0, 3, 6, -1, -5, -7, 1, 8, 7, 12, 13 ]

plt.title('line plot')
plt.grid()
plt.plot(x_data, y_data, color='b')
plt.show()
```