

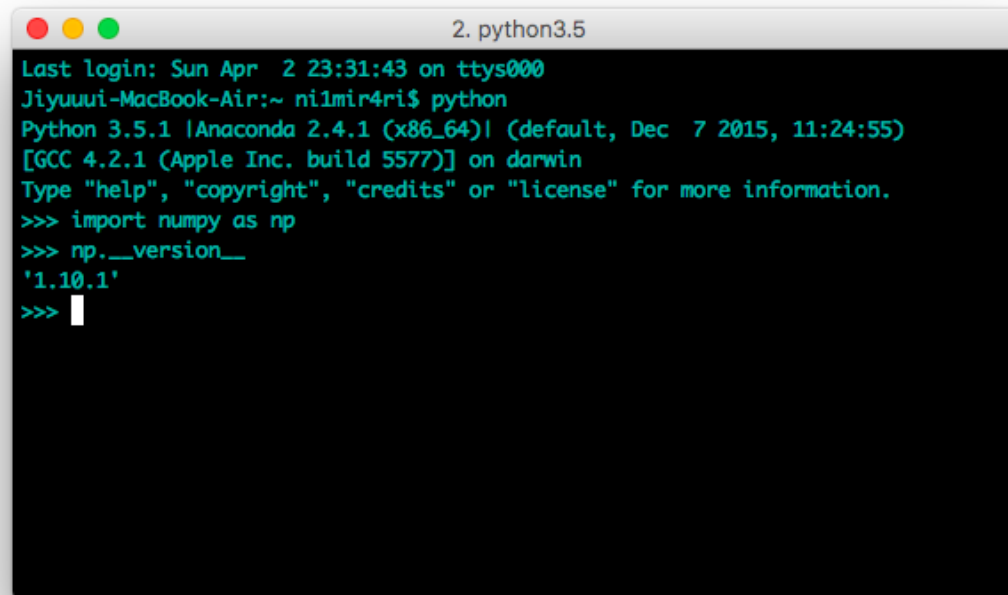
Numpy

What is Numpy

- 데이터/수치분석을 위한 python 모듈
- 행렬연산에 특화
- Matlab의 python 버전이라 생각할 수 있습니다
- Python으로 만들어지는 많은 머신러닝 모듈의 베이스가 됨

Numpy installed?

- `import numpy as np`
- `np.__version__`

A terminal window titled '2. python3.5' with a dark background and light green text. It shows the output of running 'python' in a terminal, including login information, Python version (3.5.1), and the successful execution of 'import numpy as np' and 'np.__version__', which returns '1.10.1'.

```
2. python3.5
Last login: Sun Apr  2 23:31:43 on ttys000
Jiyuuui-MacBook-Air:~ nilmir4ri$ python
Python 3.5.1 |Anaconda 2.4.1 (x86_64)| (default, Dec  7 2015, 11:24:55)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> np.__version__
'1.10.1'
>>> 
```

Numpy performance

```
performance.py
1 import numpy as np
2 import timeit
3
4 # 0~999를 담고있는 python list 만들기
5 num_list = range(1000)
6 # 0~999를 담고있는 numpy array (vector) 만들기
7 num_np_array = np.arange(1000)
8
9 # python list에 있는 모든 수의 제곱을 구함
10 start = timeit.default_timer()
11 squared_list = [i**2 for i in num_list]
12 end = timeit.default_timer()
13 # 실행시간 출력
14 print('python list: %lf 초' % (end - start))
15
16 # numpy array에 있는 모든 수의 제곱을 구함
17 start = timeit.default_timer()
18 squared_np_array = num_np_array**2
19 end = timeit.default_timer()
20 # 실행시간 출력
21 print('numpy array: %lf 초' % (end - start))
22
```

```
2. bash
Jiyuui-MacBook-Air:numpy_practice $ python performance.py
python list: 0.000473 초
numpy array: 0.000025 초
Jiyuui-MacBook-Air:numpy_practice $
```

- 같은 작업인데도 numpy array를 이용한 연산이 약 19배 빠르다

ndarray

- N-dimensional array
 - Matlab에서 사용하는 vector나 matrix로 생각할 수 있다
- 같은 종류 (type)의 데이터만 담을 수 있다
- 리스트, 튜플로 부터 ndarray 생성 가능

```
2. python (bash)
➔ ~ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> ar = np.array((10, 20, 30)) # 튜플로 ndarray 생성
>>> print(ar)
[10 20 30]
>>> print(type(ar))
<type 'numpy.ndarray'>
>>> ar2 = np.array([10, 20, 30]) # 리스트로 ndarray 생성
>>> print(ar2)
[10 20 30]
>>> print(type(ar2))
<type 'numpy.ndarray'>
>>>
```

ndarray

- 인덱스는 0부터 시작

```
2. python (bash)
>>>
>>>
>>> print(ar[0]) # 인덱스는 0부터 시작
10
>>>
```

- Python for문을 이용하여 ndarray의 각 요소들을 가져올 수 있음

```
2. python (bash)
>>> for num in ar:
...     print(num)
...
10
20
30
>>>
```

ndarray.shape

- 이중 리스트를 이용하여 2차원 행렬을 만들 수도 있음
- np.shape() 함수는 ndarray의 모양을 얻을 수 있는 함수

```
2. python (bash)
>>>
>>>
>>> list_2d = [[1,2,3],[4,5,6]]
>>> mat = np.array(list_2d)
>>> print(np.shape(mat))
(2, 3)
>>>
```

- 다차원 ndarray의 인덱싱도 python list와 똑같이 할 수 있음

```
2. python (bash)
>>> print( mat[0][0] )
1
>>> print( mat[0][1] )
2
>>> print( mat[1][0] )
4
>>>
```

ndarray.shape

- Nddarray의 shape: ndarray의 모양. 벡터 혹은 행렬의 차원
- np.reshape() 는 ndarray의 shape를 바꾸는 함수
 - 첫 번째 인자: shape를 바꿀 ndarray 변수
 - 두 번째 인자: 새로운 shape. Shape 또한 python list나 tuple로 표현

```
2. python (bash)
>>> print( np.shape(mat) )
(2, 3)
>>> flatten = np.reshape( mat, (6) )
>>> print(flatten)
[1 2 3 4 5 6]
>>> print(np.shape(flatten))
(6,)
>>> 
```


ndarray.transpose

- 머신러닝 알고리즘을 구현하다보면 복잡한 행렬 연산들을 해야함
 - 그 과정에서 차원을 바꾸는 일이 많을 수 있습니다.
- np.transpose()
 - 전치행렬 구하기

```
2. python (bash)
>>> mat
array([[1, 2, 3],
       [4, 5, 6]])
>>> mat.T
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> print( np.shape(mat) )
(2, 3)
>>> print( np.shape(mat.T) )
(3, 2)
>>>
```

```
2. python (bash)
>>> transposed_mat = np.transpose(mat)
>>> print( transposed_mat )
[[1 4]
 [2 5]
 [3 6]]
>>> print( np.shape(transposed_mat) )
(3, 2)
>>>
```

practice

0~26의 값을 갖는 3x3x3 ndarray 만들어보기

- Hint
 - range() python 내장함수 이용
 - ndarray를 만드는 np.array() 함수
 - Nddarray의 shape를 바꾸는 np.reshape() 이용
- 직접 해봅시다

ndarray.slicing

- Python slicing과 비슷하다
 - 하지만 numpy는 추가적인 기능 제공: 다차원, 값 할당
- 실습을 위한 행렬 만들기

```
2. python (bash)
>>> mat = np.reshape( np.array(range(20)), (4, 5) )
>>> mat
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> 
```

ndarray.slicing

- 특정 행만 추출하기

```
2. python (bash)
>>> mat[3]
array([15, 16, 17, 18, 19])
>>> 
```

- 몇 개의 행만 슬라이싱 하기

```
2. python (bash)
>>> mat[1:3]
array([[ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> 
```

ndarray.slicing

- 처음 행부터 몇 개만 슬라이싱

```
2. python (bash)
>>> mat[ : 3]
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> |
```

- 특정 행부터 끝까지 슬라이싱

```
2. python (bash)
>>>
>>> mat[ 2 : ]
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> |
```

ndarray.slicing

- 다차원 ndarray 인덱싱

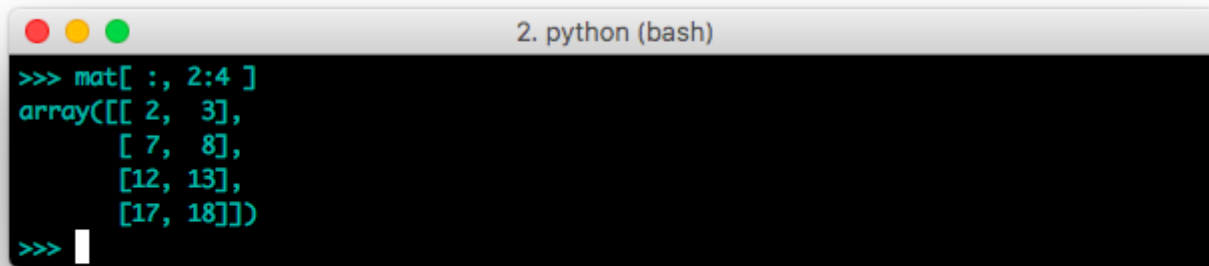
```
2. python (bash)
>>> mat[2, 1]
11
>>> 
```

- 특정 열만 슬라이싱
 - 행, 열 동시에 인덱싱 하는 것이 가능하다는 점을 이용

```
2. python (bash)
>>> mat[ : , 1]
array([ 1,  6, 11, 16])
>>> 
```

ndarray.slicing

- 몇 개의 열만 슬라이싱



```
2. python (bash)
>>> mat[ :, 2:4 ]
array([[ 2,  3],
       [ 7,  8],
       [12, 13],
       [17, 18]])
>>> 
```

- 행과 마찬가지로
 - 처음부터 특정 열까지, 특정 열부터 끝까지 슬라이싱도 가능

indexing

- Narray를 이용한 인덱싱

```
2. python (bash)
>>> indices = np.array([1, 3, 4])
>>> mat[ : , indices ]
array([[ 1,  3,  4],
       [ 6,  8,  9],
       [11, 13, 14],
       [16, 18, 19]])
>>> 
```

- Python list를 이용한 인덱싱

```
2. python (bash)
>>> python_list = [1, 2]
>>> mat[ python_list, : ]
array([[ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> 
```


Conditional indexing

- 10보다 크거나 같은 요소만 추출하기

```
2. python (bash)
>>> cond_indices = mat >= 10
>>> cond_indices
array([[False, False, False, False, False],
       [False, False, False, False, False],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]], dtype=bool)
>>> mat[cond_indices]
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
>>>
```

- 짝수인 요소만 추출해보세요.

numpy operator

- + : 전체 요소에 덧셈

```
2. python (bash)
>>> mat + 3
array([[ 3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22]])
>>>
```

- - : 전체 요소에 뺄셈

```
2. python (bash)
>>> mat - 5
array([[ -5,  -4,  -3,  -2,  -1],
       [  0,   1,   2,   3,   4],
       [  5,   6,   7,   8,   9],
       [10, 11, 12, 13, 14]])
>>>
```

numpy operator

- * : 전체 요소에 곱셈

```
2. python (bash)
>>> mat * 2
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28],
       [30, 32, 34, 36, 38]])
>>>
```

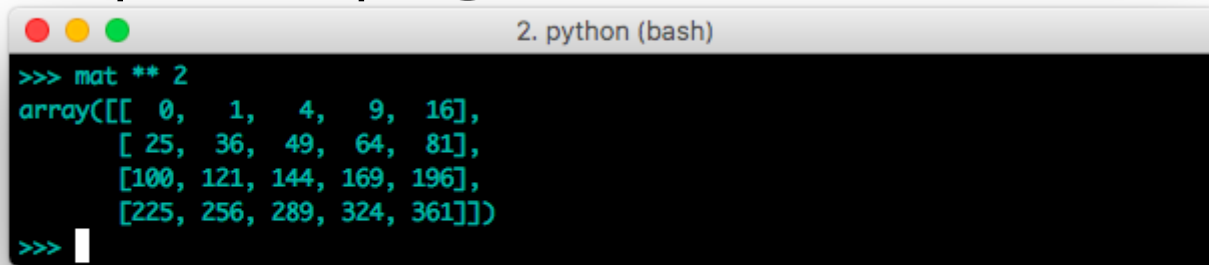
- / : 전체 요소에 나눗셈

```
2. python (bash)
>>> mat / 2
array([[0, 0, 1, 1, 2],
       [2, 3, 3, 4, 4],
       [5, 5, 6, 6, 7],
       [7, 8, 8, 9, 9]])
>>>
```

- 자료형 문제로 결과가 정확하지 않을겁니다.

numpy operator

- ** : 전체 요소에 n승

A terminal window titled "2. python (bash)" with a dark background and light green text. It shows the execution of the command 'mat ** 2' on a 4x5 numpy array. The output is a new 4x5 array where each element is the square of the corresponding element in the original array.

```
>>> mat ** 2
array([[ 0,  1,  4,  9, 16],
       [25, 36, 49, 64, 81],
       [100, 121, 144, 169, 196],
       [225, 256, 289, 324, 361]])
>>> 
```

- Python에서 사용하는 연산자는 거의 쓸 수 있습니다

Operation between elements of matrices

- 실습을 위한 두 번째 행렬 생성

```
2. python (bash)
>>> mat2 = - ( mat + 1 )
>>> mat2
array([[ -1,  -2,  -3,  -4,  -5],
       [ -6,  -7,  -8,  -9, -10],
       [-11, -12, -13, -14, -15],
       [-16, -17, -18, -19, -20]])
>>>
```

- 두 행렬의 요소별 덧셈

```
2. python (bash)
>>> elementwise_add = mat + mat2
>>> elementwise_add
array([[ -1,  -1,  -1,  -1,  -1],
       [ -1,  -1,  -1,  -1,  -1],
       [ -1,  -1,  -1,  -1,  -1],
       [ -1,  -1,  -1,  -1,  -1]])
>>>
```

Operation between elements of matrix and vector

- 실습을 위한 벡터 생성

```
2. python (bash)
>>> vector = np.array([1, 2, 3, 4, 5])
>>> 
```

- 행렬의 각 행과 벡터의 요소별 덧셈

```
2. python (bash)
>>> mat + vector
array([[ 1,  3,  5,  7,  9],
       [ 6,  8, 10, 12, 14],
       [11, 13, 15, 17, 19],
       [16, 18, 20, 22, 24]])
>>> 
```

Operation between elements of matrix and vector

- 실습을 위한 벡터 만들기

```
2. python (bash)
>>> vector = np.array([[2], [3], [4], [5]])
>>> vector
array([[2],
       [3],
       [4],
       [5]])
>>> |
```

- 행렬의 각 열과 벡터의 요소별 덧셈

```
2. python (bash)
>>> mat + vector
array([[ 2,  3,  4,  5,  6],
       [ 8,  9, 10, 11, 12],
       [14, 15, 16, 17, 18],
       [20, 21, 22, 23, 24]])
>>> |
```

- 아까 모든 요소에대한 연산과 마찬가지로 다른 연산자에도 똑같이 적용됨
- ndarray의 차원이 더 높아져도 적용됨

Multiplication of matrices

- 선형대수학에서 배웠던 행렬간의 곱셈
- `np.dot(행렬1, 행렬2)`
- 실습을 위한 두 행렬 만들기

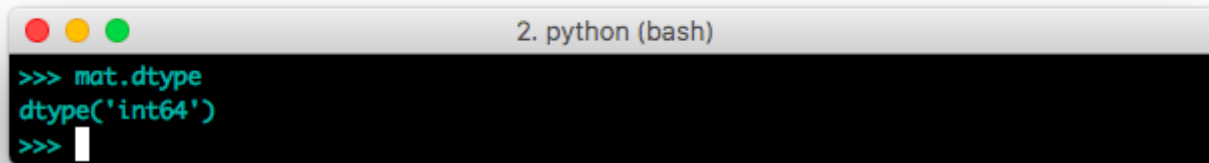
```
2. python (bash)
>>> mat1 = np.array([[1,1,1], [2,2,2]])
>>> mat2 = np.array([[1, 4], [2, 5], [3, 6]])
>>> 
```

- 두 행렬의 곱

```
2. python (bash)
>>> np.dot(mat1, mat2)
array([[ 6, 15],
       [12, 30]])
>>> 
```


Numpy data type

- ndarray.dtype

A terminal window titled "2. python (bash)" with a black background and green text. It shows the command >>> mat.dtype and the output dtype('int64').

```
>>> mat.dtype
dtype('int64')
>>> 
```

- 다양한 데이터 타입

- bool
- int
- float
- str

•

- ...
- 더 많은 정보는

https://www.tutorialspoint.com/numpy/numpy_data_types.htm

Numpy data type

- ndarray.astype(타입명)

```
2. python (bash)
>>> float_mat = mat.astype(np.float)
>>> float_mat
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.,  9.],
       [10., 11., 12., 13., 14.],
       [15., 16., 17., 18., 19.]])
>>> mat
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> 
```

Numpy data type

- ndarray.astype(타입명)
 - np.float 형으로 변경하면 나눗셈이 제대로 됩니다.

```
2. python (bash)
>>> mat/2
array([[0, 0, 1, 1, 2],
       [2, 3, 3, 4, 4],
       [5, 5, 6, 6, 7],
       [7, 8, 8, 9, 9]])
>>> float_mat/2
array([[ 0. ,  0.5,  1. ,  1.5,  2. ],
       [ 2.5,  3. ,  3.5,  4. ,  4.5],
       [ 5. ,  5.5,  6. ,  6.5,  7. ],
       [ 7.5,  8. ,  8.5,  9. ,  9.5]])
>>> 
```

Numpy data type

- ndarray.astype(타입명)
 - bool 형도 float로 변환할 수 있습니다

```
2. python (bash)
>>> bool_vec = np.array([True, True, False, True])
>>> bool_vec.dtype
dtype('bool')
>>> bool_to_float = bool_vec.astype(np.float)
>>> bool_to_float
array([ 1.,  1.,  0.,  1.])
>>>
```

Frequently used numpy functions

- `np.sum(합을 구할 행렬, axis=합을 구할 차원)`
 - 모든 요소의 합, 또는 차원별 합을 구할 수 있음

```
2. python (bash)
>>> np.sum( mat )
190
>>> 
```

- 각 행의 열에대한 합

```
2. python (bash)
>>> np.sum( mat, axis=1 )
array([10, 35, 60, 85])
>>> 
```

- 각 열의 행에대한 합

```
2. python (bash)
>>> np.sum( mat, axis=0 )
array([30, 34, 38, 42, 46])
>>> 
```

Frequently used numpy functions

- `np.mean()`(평균을 구할 행렬, `axis=`평균을 구할 차원)
 - 모든 요소의 평균, 또는 차원별 평균을 구할 수 있음

```
2. python (bash) 🛎  
>>> np.mean( mat )  
9.5  
>>> 
```

- 각 행의 열에대한 평균

```
2. python (bash)  
>>> np.mean( mat, axis=1 )  
array([ 2.,  7., 12., 17.])  
>>> 
```

- 각 열의 행에대한 평균

```
2. python (bash)  
>>> np.mean( mat, axis=0 )  
array([ 7.5,  8.5,  9.5, 10.5, 11.5])  
>>> 
```

Frequently used numpy functions

- `np.max`(최댓값을 구할 행렬, `axis`=최댓값을 구할 차원)
 - 모든 요소 중 최댓값, 또는 차원별 최댓값을 구할 수 있음

```
2. python (bash)
>>> np.max( mat, axis=1 )
array([ 4,  9, 14, 19])
>>>
```

- `np.min`(최솟값을 구할 행렬, `axis`=최솟값을 구할 차원)
 - 모든 요소 중 최댓값, 또는 차원별 최댓값을 구할 수 있음

```
2. python (bash)
>>> np.min( mat, axis=0 )
array([0, 1, 2, 3, 4])
>>>
```

Frequently used numpy functions

- `np.argmax(행렬, axis=최댓값의 인덱스를 구할)`
 - 모든 요소 중 최댓값의 인덱스

```
2. python (bash)
>>> np.argmax( mat )
19
>>>
```

- 행에서 최댓값의 인덱스

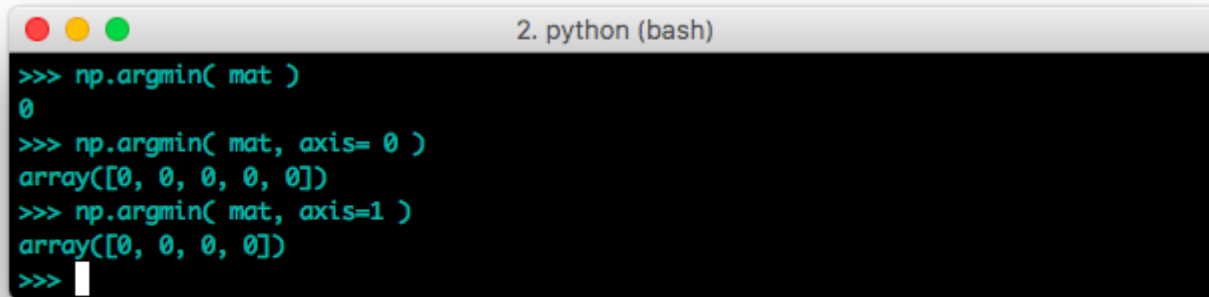
```
2. python (bash)
>>> np.argmax( mat, axis=1 )
array([4, 4, 4, 4])
>>>
```

- 열에서 최댓값의 인덱스

```
2. python (bash)
>>> np.argmax( mat, axis=0 )
array([3, 3, 3, 3, 3])
>>>
```


Frequently used numpy functions

- `np.argmin(행렬, axis=최솟값의 인덱스를 구할)`
 - `np.argmax()`와 유사

A terminal window titled "2. python (bash)" with a black background and green text. It shows the following commands and outputs:

```
>>> np.argmin( mat )
0
>>> np.argmin( mat, axis= 0 )
array([0, 0, 0, 0, 0])
>>> np.argmin( mat, axis=1 )
array([0, 0, 0, 0])
>>> 
```

Frequently used numpy functions

- `np.random.normal(평균, 표준편차, ndarray shape)`
 - 인자로 넘겨진 평균, 표준편차의 정규분포를 따르는 랜덤 값 생성
 - 머신러닝에서는 overfitting을 예방하기 위해 학습데이터에 정규분포를 따르는 잡음을 더하는 경우에 사용

```
1. python3.5
>>> np.random.normal( 0, 0.1, [2, 3] )
array([[ -0.05774687, -0.04996592, -0.12387002],
       [ -0.13395694,  0.10747827,  0.19811271]])
>>> np.random.normal( 10, 0.01, [3, 3, 2] )
array([[[ 10.00555815,  9.98903243],
        [ 9.98828184, 10.01734258],
        [ 9.99437717, 10.02196912]],

       [[ 10.00021965, 10.00248266],
        [ 9.99804544, 10.00014633],
        [ 9.99919242,  9.98977444]],

       [[ 10.0019738 ,  9.99197586],
        [ 10.00126182, 10.02242233],
        [ 9.97905301, 10.012992 ]]])
>>> |
```

Reference

- 모든 함수를 이 수업에서 다뤄볼 순 없습니다.
- 필요할 때 마다 구글링을 하세요
- Reference를 봐도 좋습니다
 - <https://docs.scipy.org/doc/numpy/reference/>

Numpy 응용해보기

- 여러 점들과 특정 점의 유클리드 거리 구하기

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

data1	3.0	1.3	7.6
data2	8.4	6.5	-32.6
data3	7.4	37.3	12.1
data4	6.9	-25.3	0.0
data5	3.2	-4.1	22.8

mean	4.3	4.2	-6.1
------	-----	-----	------

- `np.sqrt(ndarray)`: 모든 요소의 루트값을 구하는 함수

Numpy 응용해보기

- 데이터를 3개의 클래스로 분류하는 문제
 - Ont-hot encoded 예측값과 정답값을 이용하여 정확도 구해보기

예측값	class1	class2	class3
data1	0.8	0.05	0.05
data2	0.1	0.2	0.7
data3	0.6	0.4	0.0
data4	0.3	0.4	0.3
data5	0.1	0.1	0.8

정답값	class1	class2	class3
data1	1.0	0.0	0.0
data2	0.0	0.0	1.0
data3	0.0	0.0	1.0
data4	0.0	1.0	0.0
data5	1.0	0.0	0.0

Numpy 응용해보기

- 데이터를 3개의 클래스로 분류하는 문제
 - One-hot encoded 예측값과 정답값을 이용하여 cross entropy 구해보기

$$H_{y'}(y) = - \sum_i y'_i \log(y_i)$$

예측값	class1	class2	class3
data1	0.8	0.05	0.05
data2	0.1	0.2	0.7
data3	0.6	0.4	0.0
data4	0.3	0.4	0.3
data5	0.1	0.1	0.8

정답값	class1	class2	class3
data1	1.0	0.0	0.0
data2	0.0	0.0	1.0
data3	0.0	0.0	1.0
data4	0.0	1.0	0.0
data5	1.0	0.0	0.0