

Python 기초

2. Numpy



개요

Arrays 생성

Array Indexing

Data 변환과 연산

개요

- ✓ List 장점과 한계
- ✓ 목표
- ✓ 라이브러리 불러오기
- ✓ 몇가지 용어정의

List 장점과 한계

✓ 리스트

- 값의 집합(collection)
- 다른 타입의 데이터도 한꺼번에 저장 가능
- 변경, 추가, 제거 용이

✓ 그러나, 데이터 분석에서의 필요는...

- 단순히 값의 집합 개념을 넘어선 수학적 계산이 가능해야 한다.
- 대량의 데이터를 처리하는데 처리 속도가 빨라야 한다.

목표

Numpy의 모든 것을 다루지 않습니다. Machine Learning 또는 Deep Learning을 공부하기 위해 최소한 알아야 하는 범위만을 다룹니다.

1. numpy를 이용하여 다양한 모양(shape)의 행렬 array를 만들 수 있습니다.
2. numpy를 이용하여 다양한 행렬 연산을 수행할 수 있습니다.
3. 2-d array(혹은 그 이상)를 만들기.
4. 1-d array를 2-d array로 만들기.
5. 행렬 간의 곱셈.
6. csv파일 읽기
7. numpy array를 python list로 변환, python list를 numpy array로 변환
8. array shuffle, split, sampling

라이브러리 불러오기

Python

```
import numpy as np
```

```
data = np.genfromtxt("./Graduate_apply.csv", delimiter=";", names=True)  
print(type(data))
```

Python

```
import numpy
```

```
data1 = numpy.genfromtxt("./Graduate_apply.csv", delimiter=";", names=True)  
print(type(data1))
```

Python

```
from numpy import genfromtxt as ggg
```

```
data2 = ggg("./Graduate_apply.csv", delimiter=";", names=True)  
print(type(data2))
```

용어 정의

- ✓ numpy에서 각 차원을 축(**axis**)
- ✓ 축의 개수를 랭크(**rank**)
 - 예를 들어 3 X 4 행렬의 경우
 - 2차원 행렬, Rank 2 Array 라고 부름
 - 첫번째 축의 길이는 3, 두번째 축의 길이는 4
- ✓ 배열의 축 길이를 **shape**
 - 위 행렬의 shape는 (3, 4)입니다.

Arrays 생성

- ✓ 1차원, 2차원 Array
- ✓ reshape
- ✓ 여러가지 array 생성함수

1차원, 2차원 Array

✓1차원 Array

Python

```
a = np.array([1, 2, 3])
print(type(a))           # <class 'numpy.ndarray'>
print(a.shape)           # (3,)
print(a[0], a[1], a[2])  # 1 2 3


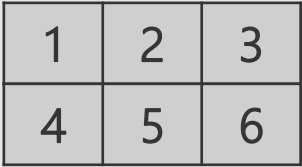
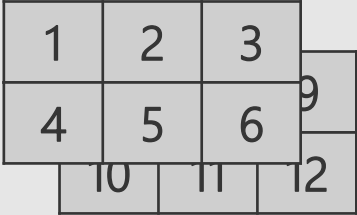
# 값 변경
a[0] = 5
print(a)                  # [5, 2, 3]
```

✓2차원 Array

Python

```
b = np.array([[1, 2, 3],
               [4, 5, 6]])
print(b.shape)           # (2,3)
print(b[0, 0], b[0, 1], b[1, 0])  # 1 2 4
```

차원 별 Array 값 및 Shape

차원	값	Shape
1차원 Array	[1, 2, 3] 	(3,)
2차원 Array	[[1, 2, 3], [4, 5, 6]] 	(2, 3)
3차원 Array	[[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]] 	(2, 2, 3)

reshape

✓ 기존 행렬을 새로운 행, 열(shape)로 다시 구성하기

✓ -1의 의미

b.reshape(3,-1) : b를 3행으로 된 array로 변환하라

b.reshape(-1,2) : b를 2열로 된 array로 변환하라

✓ 언제 쓰지?

- 이미지 분석을 위해 필요
- MNIST 흑백 이미지 5만장
shape() → (50000, 28, 28)
reshap() → (50000, 28*28)

Python

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

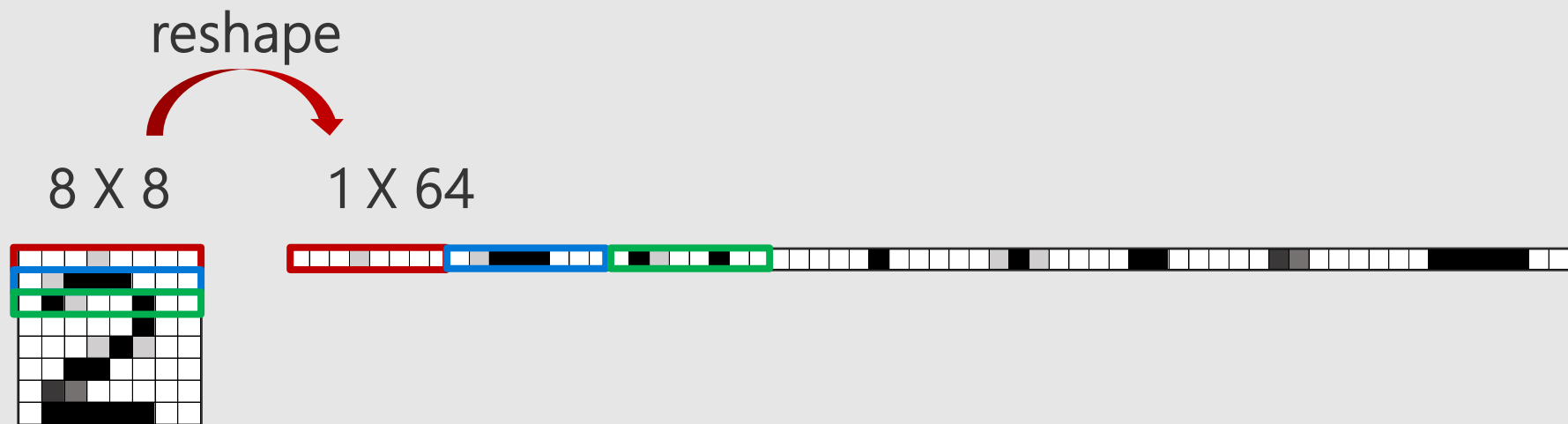
```
c = np.reshape(b, (3, 2))  
# b.reshape(3, -1)  
print(c.shape)  
print(c)
```

```
(3, 2)
```

```
[[1 2]  
 [3 4]  
 [5 6]]
```

reshape

✓ 이미지 분석



여러가지 Array 생성함수

- ✓ `np.zeros()`
 - 0으로 채워진 Array
- ✓ `np.ones()`
 - 1로 채워진 Array
- ✓ `np.full()`
 - 특정 값으로 채워진 Array
- ✓ `np.eye()`
 - 단위 행렬
- ✓ `np.random.random()`
 - 랜덤 값으로 채운 Array

Python

```
# 0으로 채워진 array 생성
a = np.zeros((2, 2))
print(a)                # "[[ 0.  0.]
                        #  [ 0.  0.]]"
```

```
# 1로 채워진 array 생성
b = np.ones((1, 2))
print(b)                # "[[ 1.  1.]]"
```

```
# 특정 값으로 채워진 array 생성
c = np.full((2, 2), 7.)
print(c)                # "[[ 7.  7.]
                        #  [ 7.  7.]]"
```

```
# 2x2 단위 행렬(identity matrix) 생성
d = np.eye(2)
print(d)                # "[[ 1.  0.]
                        #  [ 0.  1.]]"
```

```
# 랜덤값으로 채운 array 생성
e = np.random.random((2, 2))
print(e)
```

실습 #1 : Arrays 생성

Array Indexing

- ✓ Subsetting
- ✓ 데이터에 액세스 하는 두가지 방법

Subsetting

✓1차원 Array

Python

```
Score = np.array([78,91,84,89,93,65])

Score >= 90
# array([False, True, False, False, True, False], dtype=bool)

Score[Score>=90]
# array([91, 93])
```

✓2차원 Array

Python

```
Score_2d = np.array([[78,91,84,89,93,65]
                     , [82,87,96,79,91,73]])

score_2d[0] >= 90
# array([False,  True, False, False,  True, False])

score_2d[0][score_2d[0] >= 90]
# array([91, 93])

score_2d[1][score_2d[1] >= 90]
# array([96, 91])
```


np.array 데이터에 액세스 하는 방법

✓ 인덱스 번호로 가져오기

- `np_array[#, #]` : 하나의 값으로 가져옴

✓ 슬라이스로 가져오기

- `np_array[0:2, 2:4]` : 원래 차원 배열을 가져옴.

✓ 인덱스와 슬라이스를 섞으면...

- `np_array[1, 2:4]` : 더 낮은 차원의 배열이 됨.

Python

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],  
              [9, 10, 11, 12]])
```

```
# 인덱스로 가져오기
```

```
print(a[[0, 1, 2], [0, 1, 1]])
```

```
# 슬라이스만 사용하면 원래 차원 배열을 가져옴
```

```
Row_r2 = a[1:2, :] # Rank 2
```

```
Print(row_r2, row_r2.shape)
```

```
# [[5 6 7 8]] (1, 4)
```

```
# 정수 인덱스와 슬라이스를 섞으면 더 낮은 차원 배열
```

```
row_r1 = a[1, :] # Rank 1
```

```
print(row_r1, row_r1.shape) # [5 6 7 8] (4,)
```

실습 #2 : Arrays Indexing

Data 변환 및 연산

- ✓ 변환
- ✓ 기본 연산(사칙연산)
- ✓ List와 다른점
- ✓ 행렬 연산
- ✓ Shuffle, Sampling, Split

데이터 변환

✓ 숫자간 변환 : 소수를 정수로

- Array 생성시 dtype을 지정

Python

```
x = np.array([1, 2])  
print(x.dtype)           # "int64"  
  
x = np.array([1.0, 2.0])  
print(x.dtype)           # "float64"  
  
x = np.array([1.8, 2], dtype=np.int32)  
print(x.dtype)
```

✓ np.array → list

- np.array의 method로 .tolist 사용

Python

```
# Numpy Array  
x = np.array([1, 2])  
print(type(x))           # <class 'numpy.ndarray'>  
  
# numpy.ndarray to list  
y = x.tolist()           # 종종 쓰인다.  
print(y)                 # [1, 2]  
Print(type(y))           # <class 'list'>
```

기본 연산(사칙연산)

- ✓ 더하기 : +, np.add
- ✓ 빼기 : -, np.subtract
- ✓ 곱하기 : *, np.multiply
- ✓ 나누기 : /, np.divide
- ✓ 제곱근 : np.sqrt
- ✓ 제곱 : **, np.power

Python

```
x = np.array([[1, 2], [3, 4]], dtype=np.float64)
y = np.array([[5, 6], [7, 8]], dtype=np.float64)
```

```
# array 더하기
print(x + y)
print(np.add(x, y))
```

```
[[ 6.0  8.0]
 [10.0 12.0]]
```

```
# array 빼기
print(x - y)
print(np.subtract(x, y))
```

```
[[ -4.0 -4.0]
 [ -4.0 -4.0]]
```

```
# array 곱하기
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5.0 12.0]
 [21.0 32.0]]
```

```
# array 나누기
print(x / y)
print(np.divide(x, y))
```

```
[[ 0.2          0.33333333]
 [ 0.42857143  0.5        ]]
```

```
# array 제곱근
print(np.sqrt(x))
```

```
[[ 1.          1.41421356]
 [ 1.73205081  2.        ]]
```

List와 다른 점

✓ 리스트끼리 연산이 안됨

✓ 동일한 연산자에 다른 작동방식

Python

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight / height ** 2
#TypeError: unsupported operand type(s) for **

np_height = np.array(height)
np_weight = np.array(weight)
np_weight / np_height ** 2
#array([ 21.852, 20.975, 21.75 , 24.747, 21.441])
```

Python

```
python_list = [1, 2, 3]
numpy_array = np.array([1, 2, 3])

python_list + python_list
# [1, 2, 3, 1, 2, 3]

numpy_array + numpy_array
# array([2, 4, 6])
```

행렬 연산

✓행렬의 곱

- 행렬1.dot(행렬2)
- np.dot(행렬1, 행렬2)

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \text{일때,}$$
$$AB = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Python

```
x = np.array([[1, 2],  
              [3, 4]])  
y = np.array([[5, 6],  
              [7, 8]])  
  
print(x.dot(y))  
print(np.dot(x, y))
```

형렬 연산

✓Element의 합

- np.sum

✓전치행렬

- 행렬.T

Python

```
x = np.array([[1, 2],
               [3, 4]])

# 합산
print(np.sum(x))          # "10"

# 열 기준 합산
print(np.sum(x, axis=0))  # "[4 6]"

# 행 기준 합산
print(np.sum(x, axis=1))  # "[3 7]"
```

Python

```
x = np.array([[1, 2],
               [3, 4]])

print(x.T)
```


실습 #3 : 변환과 연산

Shuffle, Sampling, Split

✓ Shuffle

- 기존 데이터들 중에서 무작위로 데이터 섞기

✓ Sampling

- 임의의 데이터 추출하기
- 복원추출, 비복원 추출 : replace 옵션

✓ Split

- 데이터 분할하기

Python

```
data = np.genfromtxt("./Graduate_apply.csv",  
delimeter=",", names=True)
```

```
print(data[0:10])
```

```
# Shuffle  
np.random.shuffle(data)  
print(data[0:10])
```

```
# Sampling  
np.random.choice(data, 4)
```

```
# Split  
s1, s2, s3, s4 = data[:100], data[100:200],  
data[200:300], data[300:]  
print(len(s1))  
print(s1)  
print(len(s2))  
print(len(s3))  
print(len(s4))
```

실습 #4 : Shuffle, Sampling, Split