[2019-2 영어음성학]

이름(학번): 민지우(2016130575)

Week6 Summary

데이터를 다루기 위해서는 데이터를 벡터화해야하는데, 이는 영상, 소리, 텍스트 등 숫자회되지 않은 데이터를 숫자의 나열인 벡터 형태로 변환하는 것이다.

NumPy Basics

Creation

- 라이브러리 불러오기
  - 리스트에 숫자가 들어갈 때, 요소 간 수학적 처리를 하기 위해서는 numpy 라이브러리가 필요하다.

```
#import numpy as variable
import numpy as np
a = [1, 3, 5]
b = [2, 4, 6]
#np.array(variable)
A = np.array(a)
B = np.array(b)
a + b
>> [1, 3, 5, 2, 4, 6]
A + B
>> array([ 3,  7, 11])

#np.array( [ [  ],[  ] ] ) : [   ] 이 한 행을 의미한다.
X = np.array([[1, 2, 3], [4, 5, 6]])
>>array([[1, 2, 3],
        [4, 5, 6]])
#variable.shape
X.shape
>>(2, 3)                    #[variable 갯수] * [요소 갯수] matrix : 차원 수는 숫자 갯수
x.shape                    #2개가 있다 2*3 matrix가
>> (2, 2, 3)
```

#np.empty((배열 크기), dtype='') : 배열을 생성만 하고 특정한 값으로 초기화 하지 않는다. *empty* 명령으로 생성된 배열에는 기존 메모리에 저장되어 있던 값으므로, 배열의 원소 값을 미리 알 수 없다.

```
np.empty([2, 3], dtype='int')
>>array([[0, 0, 0],
        [0, 0, 0]])
```

#np.zeros(ones)((배열 크기), dtype='') : 크기가 정해져 있고, 모든 값이 *0*인 배열을 생성 *np.zeros(*숫자*)*는 한 행을 출력하고 *np.zeros((*숫자*1,* 숫자*2))*는 숫자*1**숫자*2 matrix*를 출력한다. 리스트를 출력하기 때문에 인덱싱하여 요솟값을 입력하는 게 가능하다. *0*이 아닌 *1*로 초기화된 배열을 생성하려면 *ones* 명령을 사용한다.

```
np.zeros((2, 3))
>>array([[0., 0., 0.],
        [0., 0., 0.]])
d = np.zeros(4)
d[0] = 1 ; d[1] =100; d[2]=-100; d[3]=-1; d
```

>>array([1., 100., -100., -1.])

+ ***#np.zeros(ones)_like(variable, dtype='') :*** 크기를 명시하지 않고, 다른 배열과 같은 크기의 배열을 생성
+ ***#x*0과 같은 결과***

#np.arange(시작, 끝(포함하지 않음), 간격, dtype='') : NumPy버전의 range 명령으로, 특정한 규칙에 따라 증가하는 수열을 만든다.

np.arange(0, 10, 2, dtype='float64') ***#0 .. n-1***

>>array([0., 2., 4., 6., 8.])

#np.linspace(시작, 끝, 갯수, dtype='') : 선형 구간을 지정한 구간의 수만큼 분할한다.

np.linspace(0, 10, 6, dtype=float)

>>array([0., 2., 4., 6., 8., 10.])

#variable.dtype: variable의 타입을 알려준다.

>>dtype('')

#variable.astype(np.바꿀dtype): variable의 dtype을 바꿔준다.

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

>>array([[1, 2, 3],
       [4, 5, 6]])

X.astype(np.float64)

>>array([[1., 2., 3.],
       [4., 5., 6.]])

x = np.array([[[1, 2, 3], [1, 2, 3]], [[3, 4, 5], [3, 4, 5]]])***#3차원 데이터 만들기 np.array([[[ ], [ ]], [[ ], [ ]]])***

#variable.ndim: variable이 n차원인지 알려준다.

x.ndim

>> 3

import matplotlib.pyplot as plt 또는 ***from matplotlib import pyplot as plt***

#np.random : 무작위 표본 추출

#np.random.normal(loc, scale, size) : 정규분포

np.random.normal(size=5)

>>array([-0.80393769, -0.27551346, 1.00166459, 1.45758192, -0.05037177])

np.random.normal(size=(2, 2, 2))

>>array([[[0.63333144, -0.26495004],
        [-0.18745881, 0.47088579]],

       [[1.0518922 , 0.41355533],
        [2.00158582, 0.15248479]]])

np.random.normal(0, 1, 10)

array([1.58356498, 0.39114895, -0.12747642, 0.10523307, -0.00723903, -0.62562718,
1.32489554, -1.08353975, -1.24026282, 0.66835564])

data = np.random.normal(0, 1, 100) ***#normal distribution을 갖는 random한 100개의 데이터***

#plt.hist(variable, bins=#) : variable에 대해 막대기 갯수가 #개인 히스토그램을 만든다. 얼마나 세세하게 보여줄 것인가에 따라 bin 갯수를 조절한다.

plt.hist(data, bins=10)

#plt.show() : 방금 만든 히스토램만 출력한다.

plt.show()

* Manipulation

#variable.reshape(배열 크기) : variable 배열 내부 데이터는 보존한 채 형태(배열 크기)만 바꾼다.

사용하는 원소의 갯수가 정해져 있기 때문에 배열 크기에 -1을 넣으면 자동으로 계산해서 크기를 맞춰준다.

```
a = np.array([[[1, 2, 3], [4, 5, 6,]], [[6,7,8], [8, 9, 10]]])
a.reshape(3, -1)
>>array([[ 1,  2,  3,  4],
        [ 5,  6,  6,  7],
        [ 8,  8,  9, 10]])
a.reshape((2,6))
>>array([[ 1,  2,  3,  4,  5,  6],
        [ 6,  7,  8,  8,  9, 10]])
np.allclose(a.reshape(-1,2), a.reshape(6,2))
>>True
```

* NumPy I/O

#numpy.random.random(/rand)(size=None): 0부터 1사이의 균일 분포의 난수를 생성한다.

```
np.random.random((2,3))
>>array([[0.49784128, 0.96587066, 0.13008896],
        [0.44370748, 0.65424579, 0.19724159]])
```

#numpy.random.randint(low, high=None, size=None): 균일 분포의 정수 난수를 생성한다.

```
np.random.randint(0,10, (2, 3))
>>array([[9, 5, 3],
        [1, 7, 1]])
```

#numpy 배열 외부 파일로 저장하고, 불러오기

#np.save(): 1개의 배열을 NumPy format의 바이너리 파일로 저장하기

#np.savez(): 여러 개의 배열을 1개의 압축되지 않은 *.npz 포맷 파일로 저장하기

```
np.savez('testt', np.random.normal(0, 10, 100), np.random.randint(0, 10, (3, 4, 2)))
```

#np.load(): np.save() 또는 np.savez()로 저장된 *.npy파일 또는 *.npz파일을 배열로 불러오기

```
npzfiles = np.load('testt.npz')
np.load('testt.npz').files
>>['arr_0', 'arr_1']
np.load('testt.npz')['arr_0']
>>열을 출력한다.
```

#ls 찾는값*: 디렉토리에서 값을 찾는다.

#ls-t 찾는값*: 디렉토리에서 값을 찾아 시간 순으로 나열한다.

이름(학번): 민지우(2016130575)

***#ls-s 찾는값**: 디렉토리에서 값을 찾아 크기 순으로 나열한다.

***#ls-a 찾는값**: 디렉토리에서 값을 찾아 .(현재 디렉토리), ..(상위 디렉토리)까지 출력한다.

***#ls-al 찾는값**: 디렉토리에서 값을 찾아 .(현재 디렉토리), ..(상위 디렉토리)까지, 좀 더 자세한 정보를 출력한다.

#del : 변수를 지운다

#%who: 변수를 확인한다.

#np.loadtxt('text.txt', delimiter = ',', skiprows = 1, dtype = 'int'): 텍스트 파일 불러오기

dtype = 입력 포맷, delimiter = 구분 기호, skiprows = 특정 행 skip, usecols = 특정 컬럼만

```
np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("red", "blue"), 'formats':('float', 'float')})
>>array([( 3.3  , 1.7  ), ( 4.4  , 2.76 ), ( 5.5  , 2.09 ), ( 6.71 , 3.19 ),
        ( 6.93 , 1.694), ( 4.168, 1.573), ( 9.779, 3.366), ( 6.182, 2.596),
        ( 7.59 , 2.53 ), ( 2.167, 1.221), ( 7.042, 2.827), (10.791, 3.465),
        ( 5.313, 1.65 ), ( 7.997, 2.904), ( 5.654, 2.42 ), ( 9.27 , 2.94 ),
        ( 3.1  , 1.3  )], dtype=[('red', '<f8'), ('blue', '<f8')])
```

#np.savetxt('text.txt', 저장할 변수, delimiter = ','): 텍스트 파일 저장하기

```
np.savetxt("regression_saved.csv", data, delimiter=",")
!ls -al regression_saved.csv
>>-rw-r--r--@ 1 minjiwoo  staff  253 11  1 16:46 regression_saved.csv
```

* Inspecting

```
arr = np.random.random([5,2,3])
print(type(arr))
>><class 'numpy.ndarray'>
print(len(arr))
>>5
print(arr.shape)
>>(5,2,3)
print(arr.ndim)
>>3
print(arr.size) #5*2*3
>>30
print(arr.dtype)
>>float64
```

* Operations

```
a = np.arange(1, 5); a
>>array([1, 2, 3, 4])
b = np.arange(9, 5, -1)
>>array([9, 8, 7, 6]
a - b
>>array([-8, -6, -4, -2])
```

#1차원을 2차원으로(arange(1,10) 요소가 총 9개니까 그 안에서 reshape 가능하다)

```
a = np.arange(1, 10).reshape(3,3)
b = np.arange(9, 0, -1).reshape(3,3)
```

#각각 원소에 대해서 비교한다. matrix 배열이 다르면 error message

```
a == b
```

>>array([[False, False, False],
[False, True, False],
[False, False, False]])

a > b

>>array([[False, False, False],
[False, False, True],
[True, True, True]])

#variable.sum(), np.sum(variable)

a.sum(), np.sum(a)

>> (45, 45)

#variable.sum(axis=0), np.sum(variable, axis=0) : 첫 번째 차원의 관점에서 실행해라 세로축 압축 (열)

a.sum(), np.sum(a)

>>(array([12, 15, 18]), array([12, 15, 18]))

#variable.sum(axis=1), np.sum(variable, axis=1) : 두 번째 차원의 관점에서 실행해라 가로축 압축 (행)

>>(array([6, 15, 24]), array([6, 15, 24]))

* Broadcasting

#한 차원만 맞으면 연산 가능하다 : 4*6과 4*1 가능