Introduction to Data Mining Lecture

Week 4: Numpy (2)

Joon Young Kim

Assistant Professor, School of AI Convergence
Sungshin Women's University

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

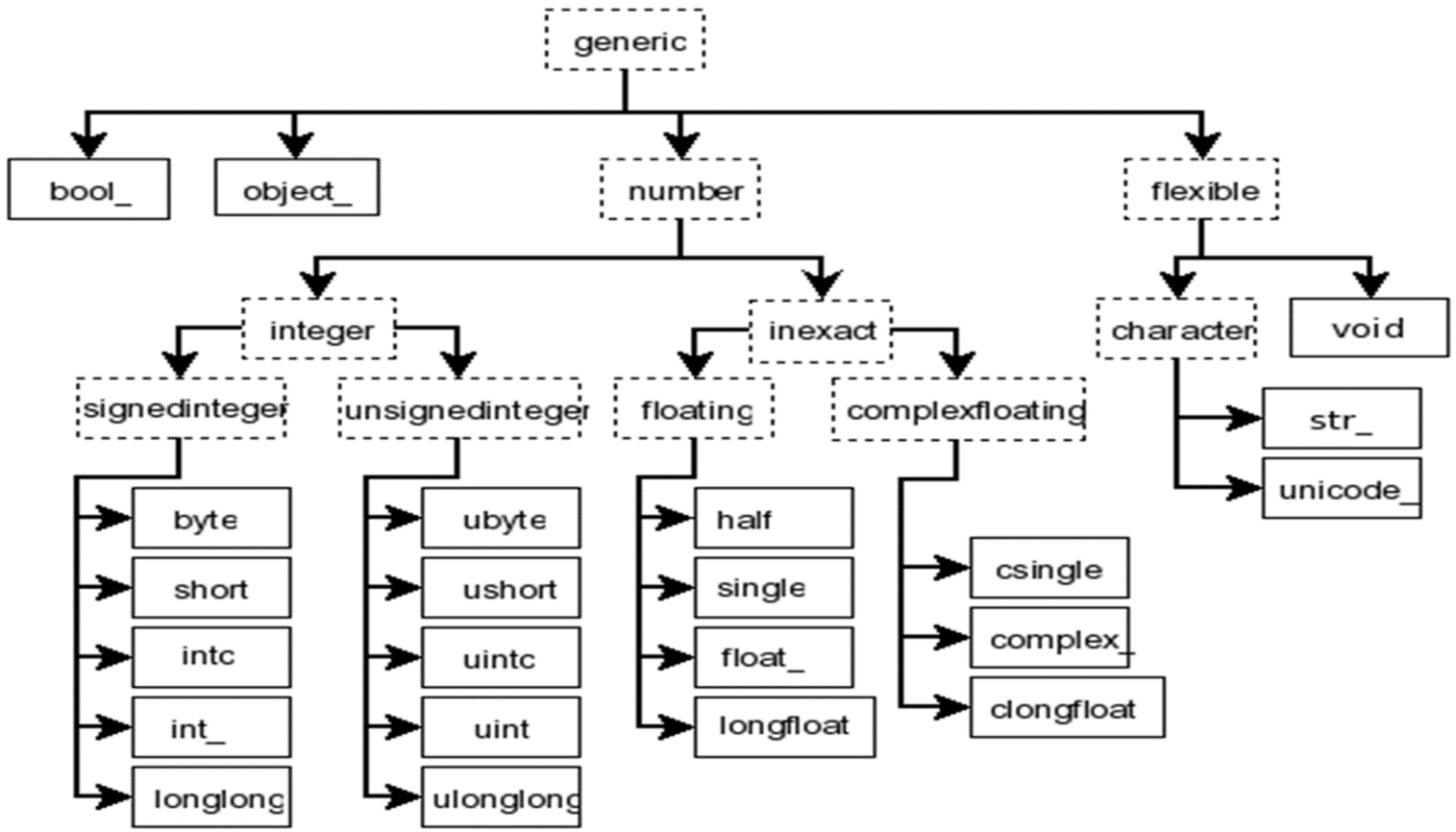
Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Week 3 Review

■ numpy 요소들에 대한 학습

→ 넘피 기본 배열, 구조화 배열 및 데이터 타잎에 대한 전반적인 학습



Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Universal Function

■ 벡터화

→ 정형화된 데이터 타입과 컴파일된 실행 프로세스

→ 실행절차를 축약하여 프로그램을 용이하게 처리

→ class numy.ufunc

속성 종류	기 능
__doc__	ufunc의 docstring
__name__	ufunc의 name
ufunc.nin	입력의 수
ufunc.nout	출력의 수
ufunc.nargs	인수(argument)의 수
ufunc.ntypes	타입의 수
ufunc.types	입력에서 출력으로 그룹화된 타입의 리스트를 반환
ufunc.identity	identity 값
ufunc.signature	일반화된 ufunc가 연산하는 중심 요소의 정의

Universal Function

■ 벡터화

→ 정형화된 데이터 타입과 컴파일된 실행 프로세스

→ 실행절차를 축약하여 프로그램을 용이하게 처리

→ class numy.ufunc

np.add.nin
np.add.nout
np.exp.nin
np.exp.nout

Universal Function

■ 벡터화

→ 정형화된 데이터 타입과 컴파일된 실행 프로세스

→ 실행절차를 축약하여 프로그램을 용이하게 처리

→ ufunc의 메소드

메서드 종류	기능 설명
reduce(a[, axis, dtype, out, keepdims])	한 축을 따라 ufunc를 적용하여 1개씩 a의 차원을 축소시킨다.
accumulate(array[, axis, dtype, out])	모든 요소에 연산자를 적용시킨 결과를 축적한다.
reduceat(a, indices[, axis, dtype, out])	단일 축에 대해 특정 자르기를 가지는 reduce를 실행한다.
outer(A,B, **kwargs)	A에 a, B에 b인 모든 쌍(a, b)에 ufunc op를 적용시킨다.
at(a, indices[, b])	색인에 의해 명시된 요소에 대해 연산 대상인 ‘a’에 제 위치에서 unbuffered 연산을 실행시킨다.

Universal Function

- 벡터화

  → 정형화된 데이터 타입과 컴파일된 실행 프로세스

  → 실행절차를 축약하여 프로그램을 용이하게 처리

  → ufuncd의 메소드

```
np.add([1,2,3,4,5], [1,2,3,4,5])
arr = np.array([1, 2, 3, 4, 5])
np.add.reduce(arr)
```

Universal Function

■ 1차원 이상의 배열에 대해 적용시 감소가 axis0에 대해 실행

```
arr = np.array(([1, 2, 3], [4, 5, 6]))
arr
np.add.reduce(arr)
np.add.reduce(arr, axis=1)
```

■ 메소드 accumulate의 적용

```
np.add.accumulate([1, 2, 3, 4, 5])
np.multiply.accumulate([1, 2, 3, 4, 5])

arr = np.arange(12).reshape((3, 4))
np.add.accumulate(arr)
np.add.accumulate(arr, axis=1)
```

Universal Function

■ reduceat의 적용

```
arr = np.arange(7)
arr
np.add.reduceat(arr, [0, 3, 5, 6])
```

```
np.linspace(0, 15, 16)
arr = np.linspace(0, 15, 16).reshape(4, 4)
arr
np.add.reduceat(arr, [0, 3, 1, 2, 0])
```

■ outer의 적용

```
np.multiply.outer([1, 2, 3], [4, 5, 6])
```

유용한 ufuncs

- numpy.subtract

```
np.subtract(1.0, 4.0)
arr1 = np.arange(9.0).reshape((3, 3))
arr2 = np.arange(3.0)
np.subtract(arr1, arr2)
```

- numpy.power

```
arr1 = range(6)
arr1
np.power(arr1, 3)
arr2 = [1.0, 2.0, 3.0, 3.0, 2.0, 1.0]
np.power(arr1, arr2)
```

유용한 ufuncs

■ numpy.sin

```
np.sin(np.pi/2)
np.sin(np.deg2rad(90))
np.sin(np.array((0., 30., 45., 60., 90.)) * np.pi/180.)
```

■ sin 함수 플롯

```
import numpy as np
import matplotlib.pyplot as plt
arr = np.linspace(-np.pi, np.pi, 201)
plt.plot(arr, np.sin(arr))
plt.xlabel('Angle [rad]')
plt.ylabel('sin(x)')
plt.axis('tight')
plt.show()
```

유용한 ufuncs

■ binary 연산

```
np.bitwise_and(13, 17)
np.binary_repr(12)
np.bitwise_and([14, 3], 13)
```

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

View/Copy

- 뷰(view)

  → 뷰는 2객체가 공유된다는 것을 의미

  → 뷰는 배열의 변화를 반영 – 메모리를 다시 인터프리팅

  → 뷰의 생성방법은 slice view나 dtype views로 생성

- slice view

```
arr = np.arange(10)
arr
v1 = arr[1:2]
arr[1] = 2
v2 = arr[1::3]
arr[7] = 10
```

View/Copy

■ dtype view

→ dtype를 할당

```
arr = np.zeros(2, dtype=np.uint16)
arr
arr.view(np.uint8)
arr.view(np.uint32)
```

■ astype을 이용한 복사

```
arr = np.zeros(2, dtype=np.uint16)
arr
arr.astype(np.unit8)
arr.astype(np.uint32)
```

View/Copy

■ 2차원 배열의 viewing

```
arr = np.arange(4, dtype=np.uint16).reshape(2, 2)
arr
arr.view(np.unit8)
arr * 100
(arr * 100).view(np.uint8)
```

■ 배열 요소의 타입이 작은 타입 ↔ 큰 타입으로 변경

```
arr = np.arange(10, dtype='int16')
arr
v1 = arr.view('int32')
v1 += 1
v2 = arr.view('int8')
```

View/Copy

- strides가 (4, 2)

```
arr = np.arange(4, dtype=np.uint16).reshape(2, 2)
arr
arr.strides
```

View/Copy

■ 복사(copy)

→ arr에서 불리언 색인처리는 복사를 반환

→ 반환된 복사본의 요소를 변경시켜도 원본 arr의 변화가 없다.

```
arr = np.random.randn(3, 4)
arr
arr1 = arr[arr>0]
arr1
arr1[:] = 0
arr
```

View/Copy

- 원본 arr에 자르기를 하여 다른 값으로 동적할당하면 원본의 요소가 변경

```
arr1 = arr[0, :]
arr1
arr1[:] = 0
arr
```

View/Copy

■ Copy함수의 적용

```
arr = np.random.randn(3, 4)
arr
arr1 = arr[0, :]; arr1
arr2 = arr1[:].copy()
arr2[:] = 0
arr1
```

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

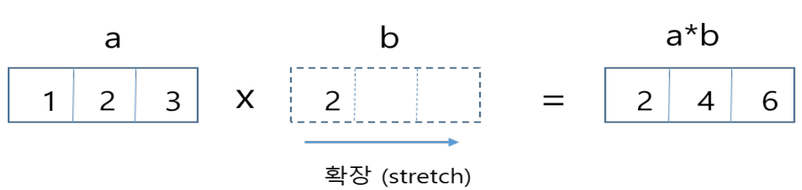
Broadcasting

- 다른 shape를 가지는 배열들을 산술 연산
- 코드의 단순화와 줄임

```
a = np.array([1.0, 2.0, 3.0])
b = np.array([2.0, 2.0, 2.0])
arr = a * b
arr
```

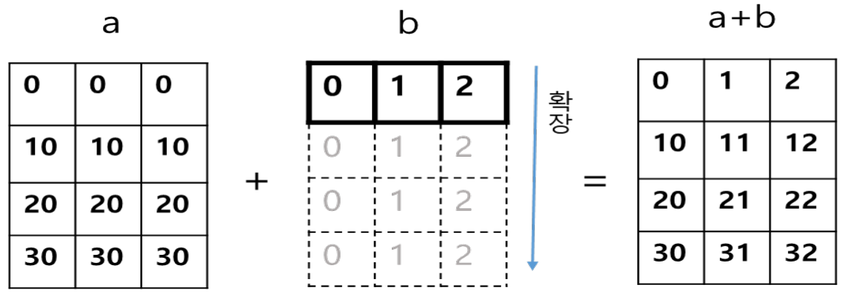
- 배열과 스칼라의 브로드캐스팅

```
a = np.array([1.0, 2.0, 3.0])
b = 2.0
a * b
```



Broadcasting

■ Shape (4, 3)인 배열 a와 shape (3,)인 배열b의 덧셈연산



■ 브로드캐스팅의 일반적인 규칙

이미지 (3차원 배열) : 256 x 256 x 3
조정(scale) (1차원 배열) : 3
조정 결과 (3차원 배열) : 256 x 256 x 3

Broadcasting

■ 비교되는 차원들 중의 하나가 1이면 다른 차원이 사용된다.

A (4차원 배열) : 8 x 1 x 6 x 1
B (3차원 배열) : 7 x 1 x 5
결과 (4차원 배열) : 8 x 7 x 6 x 5

■ A가 3차원, B가 3차원 배열

A (3차원 배열) : 15 x 3 x 5
B (3차원 배열) : 15 x 1 x 5
결과 (3차원 배열) : 15 x 3 x 5

Broadcasting

■ 브로드캐스팅 규칙의 적용

```
arr1 = np.arange(4)
arr3 = np.ones(5)
arr1.shape
arr1 + arr3
arr2.shape
arr2 + arr3
arr4.shape
arr1 + arr4
```

```
arr2 = arr1.reshape(4, 1)
arr4 = np.ones((3, 4))
arr3.shape
(arr2 + arr3).shape
(arr1 + arr4).shape
```

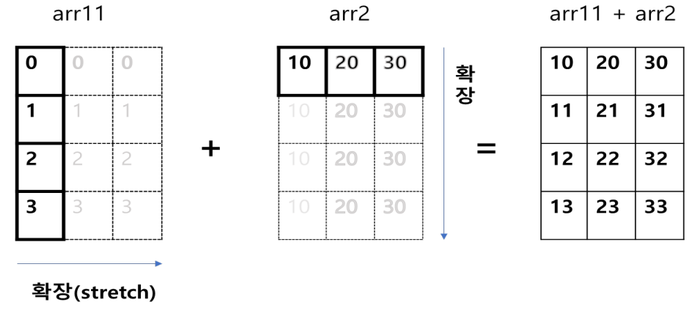
Broadcasting

- numpy.newaxis의 적용
- 1차원 배열을 2차원의 열벡터나 행벡터로 변환

```
arr = np.array([0, 1, 2, 3])
arr.shape
arr[np.newaxis, :]
arr[:, np.newaxis]
```

- shape (4, 1)과 shape (3,)의 연산

```
arr1 = np.array([0, 1, 2, 3])
arr2 = np.array([10, 20, 30])
arr11 = arr1[:, np.newaxis]
arr11 + arr2
```



Broadcasting

- broadcast 클래스
- 브로드캐스팅을 모방하는 객체 생성
- 입력 매개변수들을 차례로 브로드캐스팅하고 결과를 쌓는 객체 반환

클래스	numpy.broadcast
매개변수	in1, in2, ... : 유사배열, 입력 매개변수
반환값	b : 브로드캐스트 객체

구 분	종 류	기 능
속성	index	브로드캐스트 된 결과에서 현재 색인
	iters	self의 요소에 따르는 반복자(iterators)의 튜플
	nd	브로드캐스트 된 결과의 차원 수
	ndim	브로드캐스트 된 결과의 차원 수(넘파이 1.12.0 이후)
	numiter	브로드캐스트 된 결과에 의해 소유된 반복자의 수
	shape	브로드캐스트 된 결과의 shape
	size	브로드캐스트 된 결과의 총 크기
메서드	reset()	브로드캐스트 된 결과의 iterators를 리셋

Broadcasting

■ 속성 numiter

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([[4], [5], [6]])
arr = np.broadcast(arr1, arr2)
arr.numiter
```

■ shape (3, 1)과 shape (3,)인 배열의 연산

```
arr1 = np.array([[1], [2], [3]])
arr2 = np.array([4, 5, 6])
arr = np.broadcast(arr1, arr2)
arr
arr.shape
```

Broadcasting

■ flatiter 객체 생성, arr1+arr2 연산

→ numpy.ndarray.flat은 1-D iterator로서 numpy.flatiter 인스턴스

→ 클래스 Numpy.ndarray.flatiter는 배열에 대해 반복 처리하기 위한 flat iterator 객체

→ 배열 x에 대하여 x.flat에 의해 반환된 것이 flatiter iterator

```
out = np.empty(arr.shape)
out.flat = [u+v for (u, v) in arr]
type(out.flat)
numpy.flatiter
out                              arr1 + arr2
```

```
arr = np.arange(6).reshape(2, 3)
fl = arr.flat                    type(fl)
for item in fl:                   f1[2:4]
...     print(item)
```

■ flatiter 클래스의 속성과 메서드

구분	종류	기능
속성	base	반복되는 대상인 배열을 참조(reference)
	coords	현재 좌표(coordinates)의 N차원 튜플
	index	배열로의 flat 색인
메서드	copy()	1차원 배열로서 iterator의 복사본을 얻는다.

Broadcasting

```
arr = np.arange(6).reshape(2, 3)
fl = arr.flat                    type(fl)
for item in fl:                   f1[2:4]
...     print(item)
```

■ 속성 base와 index의 사용

```
arr1 = np.arange(5)          fl = arr1.flat
fl.base is arr1
arr2 = np.arange(6).reshape(2, 3)
f2 = arr2.flat                  f2.index
```

Broadcasting

■ numpy.flatiter.coords – 현재 좌표를 N차원의 튜플로 표기

```
arr1 = np.arange(6).reshape(2, 3)
fl = arr1.flat                    fl.coords
next(fl)                          f1.coords
next(fl)                          f1.coords
```

■ ndarray의 속성 flat

```
arr = np.arange(1, 7).reshape(2, 3)
arr                               arr.flat[3]
arr.T                             arr.T.flat[3]
type(arr.flat)
```

Broadcasting

■ 속성 flat의 적용 및 동적할당

arr.flat[[1, 4]]
arr.flat[[1, 4]] = 1; arr

arr.flat = 3; arr

Broadcasting

■ 브로드캐스팅 연산

→ 입력 데이터보다 더 많은 출력 데이터를 얻기

```
arr = np.arange(10)
arr_br = arr – arr[:, np.newaxis]
arr_br
```

Broadcasting

■ 브로드캐스팅의 이미지프로세싱 및 관리의 적용

```
import numpy as np
import matplotlib.pyplot as plt
arr1 = np.arange(10)
arr2 = np.arange(7)
arr_img = np.sqrt(arr1[:, np.newaxis]**2 + arr2**2)
plt.pcolor(arr_img)
plt.colorbar()
plt.axis('equal')
plt.show()
```

Broadcasting

■ 우리나라 수능고사의 학력 분석 예시

→ 국어, 영어, 수학 점수가 정규분포를 따른다.

→ 각각의 평균 정수는 70, 67, 59점이고 표준편차는 5, 7, 9점이고 샘플은 15개

```
arr = np.random.normal(loc=[70., 67., 59.],
                       scale = [5., 7., 9.], size = (5, 3))
arr
```

Broadcasting

■ shape (5, 3)인 arr에서 shape (3,)인 arr_avc를 뺀다.

```
arr_avc = arr.mean(axis = 0)
arr_avc
arr – arr_avc
arr_avr = arr.mean(axis = 1)
arr_avr
```

Broadcasting

- 연산이 가능하도록 shape (5,)인 arr_avr의 차원을 (5, 1)로 늘려준다.

arr_avr[:, np.newaxis]

arr – arr_avr[:, np.newaxis]

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Array Manipulation and Alignment

■ C우선과 F우선의 배치

→ 넘파이 배열은 행우선(c-major)으로 처리된다.

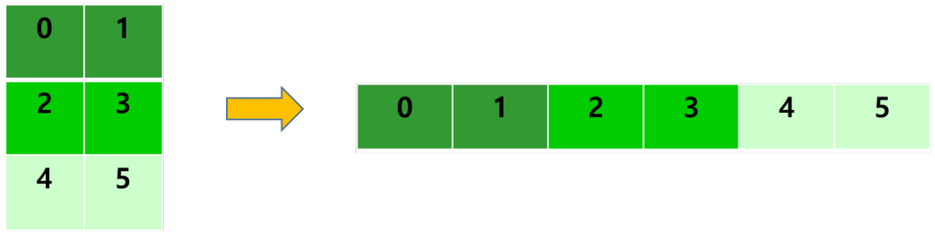
```
arr = np.arange(12).reshape(3,4)
arr
arr.ravel(order='F')
arr.ravel(order='C')
```

→ ravel함수는 1차원의 flattened array를 반환

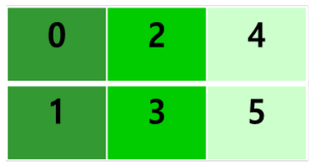
```
arr.reshape(2, 6)
arr.ravel(order='F').reshape(2, 6)
```

Array Manipulation and Alignment

■ 배열 arr의 shape (3,2)의 표기와 C우선 메모리 배치



■ arr을 전치하면 arr.T가 되고 Fortran인접배열이 된다.



Array Manipulation and Alignment

- C인접 배열이 F인접배열보다 연산이 빠르며 넘파이는 디폴트로 C인접배열을 연산

```
arr = np.arange(6)
np.reshape(arr, (2, 3)) # C-like index ordering
np.reshape(np.ravel(arr), (2, 3)) # C ravel 다음 C reshape
np.reshape(arr, (2,3), order='F') # Fortran-like index ordering
np.reshape(np.ravel(arr, order='F'), (2, 3), order='F')
```

Array Manipulation and Alignment

■ 배열을 연접하고 스택으로 배치하기

→ numpy.concatenate 함수의 적용

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6]])
np.concatenate((arr1, arr2), axis=0)
np.concatenate((arr1, arr2.T), axis=1)
np.concatenate((arr1, arr2), axis=None)
```

→ numpy.vstack 및 numpy.hstack 함수의 적용

```
np.hstack((arr1, arr2))
np.vstack((arr1, arr2))
```

Array Manipulation and Alignment

■ numpy.stack 함수의 적용

```
arrays = [np.random.randn(3, 4) for _ in range(10)]
np.stack(arrays, axis=0).shape
np.stack(arrays, axis=1).shape
np.stack(arrays, axis=2).shape
arr1 = np.array([1, 2, 3])
arr2 = np.array([2, 3, 4])
np.stack((arr1, arr2))
np.stack((arr1, arr2), axis=-1)
```

Array Manipulation and Alignment

■ 배열의 정렬

→ 정렬 구분

```
arr = np.array([[1, 4], [3, 1]])
np.sort(arr) # 마지막 축을 따라 정렬
np.sort(arr, axis=None)  # 풀려진 배열을 정렬
np.sort(arr, axis=0)     # 첫 번째 축을 따라 정렬
```

Array Manipulation and Alignment

■ 구조화배열 정렬 시 order 키워드를 'height' 사용

```
dtype = [('name', 'S10'), ('height', float), ('age', int)]
values = [('Jin', 175, 59), ('Suho', 185, 19),('Naeun', 162, 28)]
arr = np.array(values, dtype=dtype)   # 구조화된 배열 생성
np.sort(arr, order='height')
```

■ 구조화배열 정렬 시 order 키워드를 age, height 사용

```
np.sort(arr, order=['age', 'height'])
```

Array Manipulation and Alignment

■ numpy.ndarray.sort에 의한 정렬

```
arr = np.array([[1, 4], [3, 1]])
arr.sort(axis=1)                    arr
arr.sort(axis=0)                    arr
```

■ 1차원 배열의 numpy.argsort 연산

```
arr = np.array([3, 1, 2])
sorted_by_index = np.argsort(arr)
sorted_by_index
for i in sorted_by_index:
...     print(arr[i])
```

Array Manipulation and Alignment

■ 2차원 배열의 numpy.argsort 연산

```
arr = np.array([[0, 3], [2, 2]])
arr
np.argsort(arr, axis=0)    # 첫 번째 축을 따라 정렬
np.argsort(arr, axis=1)    # 마지막 축을 따라 정렬
```

Array Manipulation and Alignment

■ keys로 정렬하는 연산

```
arr=np.array([(1,0), (0,1)], dtype=[('x','<i4'), ('y','<i4')])
arr
np.argsort(arr)
np.argsort(arr, order=('x', 'y'))
np.argsort(arr, order=('y', 'x'))
```

■ numpy.lexsort의 연산

```
arr1 = np.array([2, 2, 1, 1])
arr2 = np.array([2, 2, 1, 1])
arr = np.lexsort((arr1, arr2))
arr
```

Array Manipulation and Alignment

■ 첫 번째로 성, 그 다음은 이름으로 정렬

```
surnames = (‘Hong’, ‘Na’, ‘Kim’)
first_names = (‘Gildong’, ‘Haeseok’, ‘Mandeok’)
ind = np.lexsort((first_names, surnames))
ind
[surnames[i] + “,” + first_names[i] for i in ind]
```

Array Manipulation and Alignment

■ 숫자들로 구성된 a와 b의 정렬

```
a = [1, 5, 1, 4, 3, 4, 4]      # 첫 번째 열
b = [9, 4, 0, 4, 0, 2, 1]      # 두 번째 열
ind = np.lexsort((b, a))        # a에 의해 정렬한 다음 b에 의해 정렬
print(ind)
[(a[i], b[i]) for i in ind]
[(a[i], b[i]) for i in np.argsort(a)]
```

■ argsort에 의해 정렬되는 구조화 배열

```
arr = np.array([(1, 9), (5, 4), (1, 0), (4, 4),(3, 0), (4,2), (4, 1)],
                    dtype = np.dtype([('x', int), ('y', int)]))
np.argsort(arr)      # 또는 np.argsort(arr, order=('x', 'y'))
```

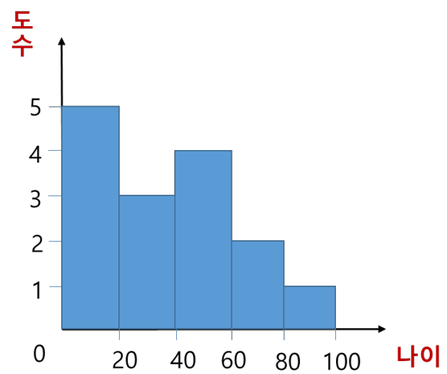
Array Manipulation and Alignment

■ np.searchsorted – 삽입되어야 하는 위치에서 순서에 따른 색인

```
np.searchsorted([1, 2, 3, 4, 5], 3)
np.serachsorted([1, 2, 3, 4, 5], 3, side='right')
np.searchsorted([1, 2, 3, 4, 5], [-10, 10, 2, 3])
```

■ 히스토그램 함수 및 예

Bin	도수(Frequency)	Bin에서 나이 분포
0 – 20	5	15, 16, 16, 17, 19
20 – 40	3	20, 22, 35
40 – 60	4	43, 45, 55, 59
60 – 80	2	60, 75
80–100	1	88



Array Manipulation and Alignment

■ 히스토그램 도수분포 와 bin

```
arr = np.array([15,16,16,17,19,20,22,35,43,45,55,59,60,75,88])
np.histogram(arr, bins=[0,20,40,60,80,100])
a, b = np.histogram(arr, bins=[0,20,40,60,80,100])
```

■ matplotlib에 의한 플롯

```
import matplotlib.pyplot as plt
import numpy  as np
arr = np.array([15,16,16,17,19,20,22,35,43,45,55,59,60,75,88])
plt.hist(arr, bins=[0, 20, 40, 60, 80, 100])
plt.title(“numbers depending on ages”)
plt.show()
```

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Array Operation

■ 다차원 배열 연산

→ 2차원 배열 연산

```
arr = np.array([[1, 2, 3], [4, 5, 6]], dtype='int32')
arr
arr.sum(axis=0)
arr.sum(axis=1)
```

Array Operation

■ 3차원 배열 연산

arr = np.arange(24).reshape(3, 2, 4)
arr
arr.shape

arr.sum(axis=0)

arr.sum(axis=1)

arr.sum(axis=2)

Array Operation

- random 함수의 리스트 생성과 넘파이의 배열 연산의 실행 속도

```
IPython: C:Users/User

(base) C:₩Users₩User>ipython
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import random

In [2]: import numpy as np

In [3]: n = 1000000

In [4]: a = [random.random() for i in range(n)]

In [5]: b = [random.random() for i in range(n)]

In [6]: arr1 = np.array(a)

In [7]: arr2 = np.array(b)

In [8]: %timeit [a[j] + b[j] for j in range(n)]
187 ms ± 624 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [9]: %timeit arr1 + arr2
5.66 ms ± 584 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [10]: %timeit np.sum(arr1)
1.34 ms ± 5.69 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [11]: %timeit sum(a)
5.38 ms ± 88.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [12]:
```

Array Operation

■ 배열 반복(Array Iteration)

→ 1차원 배열

```
arr = np.array([1, 3, 5, 7])
for i in arr:
...     print(i)
```

→ 2차원 배열

```
arr = np.arange(8).reshape(4, 2)
for i in arr:
...     print(i)
```

→ 곱셈 연산 배열

```
arr = np.arange(8).reshape(4, 2)
for (a, b) in arr:
...     print(a*b)
```

Array Operation

■ 임의 수의 연산(random number generator)

→ random 모듈의 rand 함수

np.random.rand(3, 2)

→ random 모듈의 randint 함수

np.random.randint(2, size=10)
np.random.randint(5, size=10)
np.random.randint(5, size=(2, 4))

Array Operation

■ random 모듈의 choice 함수

```
np.random.choice(5, 3)
np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0])
arr = np.arange(12)
np.random.choice(arr, size=10, replace=True)
np.random.choice(arr, size=10, replace=False)
np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0]
```

```
arr_like = [‘yakyong’, ‘jegong’, ‘manduck’, ‘haeseok’]
np.random.choice(arr_like, 5, p=[0.5, 0.1, 0.1, 0.3])
```

Array Operation

■ 선형 대수(linear algebra)

→ numpy.linalg 모듈

→ 인공지능과 딥러닝에서 사용

→ ax = b에서 x구하기

```
a = np.array([[1, 2, 3], [2, 0, 1], [1, -1, 1]])
c = np.array([[-2, 4, -1]])
b = c.T
x = np.linalg.solve(a, b)
x
```

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} -2 \\ 4 \\ -1 \end{bmatrix}$$

Array Operation

■ 3*x0 + x1 = 9와 x0 + 2*x1 = 8을 푸는 예제

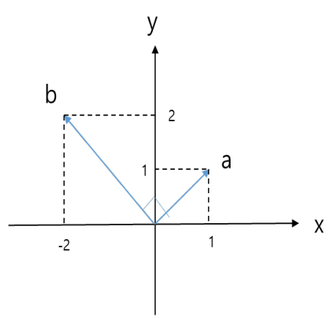
```
a = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])
x = np.linalg.solve(a, b)
x
```

■ arr1, arr2의 dot product

→ 각각의 배열 내부 요소를 곱해서 더한다.

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

■ dot product



첫 번째 방법:

$a \cdot b = |a| \times |b| \times \cos(\theta)$ 에서 θ가 90도 이므로 0

두 번째 방법:

$a \cdot b = a_x b_x + a_y b_y = 1x(-2) + 1x2 = 0$

Array Operation

- 2차원 이상의 배열 arr1과 arr2의 dot product는 arr1의 마지막 축과 arr2의 두 번째 마지막 축에 대해 곱해서 더한다.

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[4, 5], [6, 7]])
np.dot(arr1, arr2)
```

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Input/Output

■ 3가지 입출력 방법

→ 파이썬 고유의 내장함수로 처리 : open(), read(), write(), close()

→ 넘파이에서 제공하는 방법

→ 판다스에서 제공하는 방법 : read_csv(), to_csv() 등

■ NumPy binary 파일(NPY, NPZ)

→ NPY포맷 – 표준 2진파일로서 파일에 대한 정보와 함께 저장장치에 넘파이 배열들을 저장

→ shape와 dtype 정보를 저장

→ NPZ포맷 – 복수의 넘파이 파일들을 유지하기 위한 표준 포맷

→ 복수의 NPY파일들을 포함하는 하나의 zip파일

Input/Output

■ 제공 함수

함수의 종류	기능 설명
load(file[, mmap_mode, allow_pickle, ...])	.npy, .npz 또는 pickled 파일들로부터 배열이나 pickled 객체들을 로딩한다.
save(file, arr[, allow_pickle, fix_imports])	넘파이 .npy 포맷으로 2진파일을 저장한다.
savez(file, *args, **kwds)	압축되지 않는 .npz포맷으로 몇몇 배열들을 1개의 파일로 저장한다.
savez_compressed(file, *args, **kwds)	압축된 .npz포맷으로 몇몇 배열들을 1개의 파일로 저장한다.

■ 데이터의 생성과 numpy.save 및 numpy.load의 사용

```
arr = np.arange(12).reshape(3, 4)
np.save('svd_arr', arr)
np.load('svd_arr.npy')
```

Input/Output

■ numpy.savez함수의 사용

```
arr1 = np.arange(12).reshape(3, 4)
arr2 = arr1 * 1.1
np.savez('svdz_arr', a=arr2, b=arr1)
np.load('svdz_arr.npz')
arr_zp = np.load('svdz_arr.npz')
arr_zp['a']
arr_zp['b']
```

Input/Output

■ 텍스트 파일 제공 함수

함수의 종류	기능 설명
loadtxt(fname[, dtype, comments delimiter, ...])	텍스트 파일로 부터 데이터를 로딩한다.
savetxt(fname, X[, fmt, delimiter, newline, ...])	배열을 테스트 파일로 저장한다
genfromtxt(fname[, dtype, comments, ...])	지정된 대로 취급된 손실 값들을 가진 채로 텍스트 파일로부터 데이터를 로딩한다.
fromregex(file, regexp, dtype[, encoding])	정규 표현식(regular expression) 파싱을 사용하여 텍스트 파일로부터 배열을 생성한다.
fromstring(string[, dtype, count, sep])	문자열(string)로 된 텍스트 데이터로부터 초기화 된 새로운 1차원 배열
ndarray.tofile(fid[, sep, format])	배열을 텍스트나 2진(디폴트)로 파일에 쓴다.
ndarray.tolist()	리스트(아마 nested형태)로 배열을 반환

Input/Output

■ StringIO – 파일 객체로 데이터를 처리

```
import numpy as np
from io import StringIO
x = StringIO(“0 1 2₩n3 4 5”)
np.loadtxt(x)
```

■ 구분자의 사용

```
import numpy as np
from io import StringIO
x = StringIO(“0, 1, 2₩n3, 4, 5”)
np.loadtxt(x, delimiter=’,’)
```

Input/Output

■ 배열을 텍스트 파일로 저장 (CSV 파일 형태 저장도 가능)

```
arr1 = arr2 = arr3 = np.arange(0.0, 5.0, 1.0)
np.savetxt('test1.txt', arr1, delimiter=',') # arr1은 배열
np.savetxt('test2.txt', (arr1,arr2,arr3)) # 동일 크기의 2D 배열
np.savetxt('test3.txt', arr1, fmt='%1.4e') # 지수 표기
np.loadtxt('test1.txt')
np.loadtxt('test2.txt')
np.loadtxt('test3.txt')
```

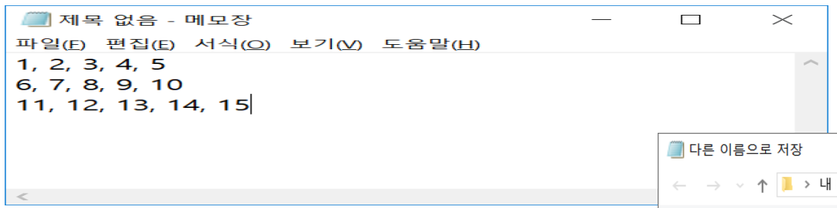
■ test3.txt가 지수 형태임을 확인

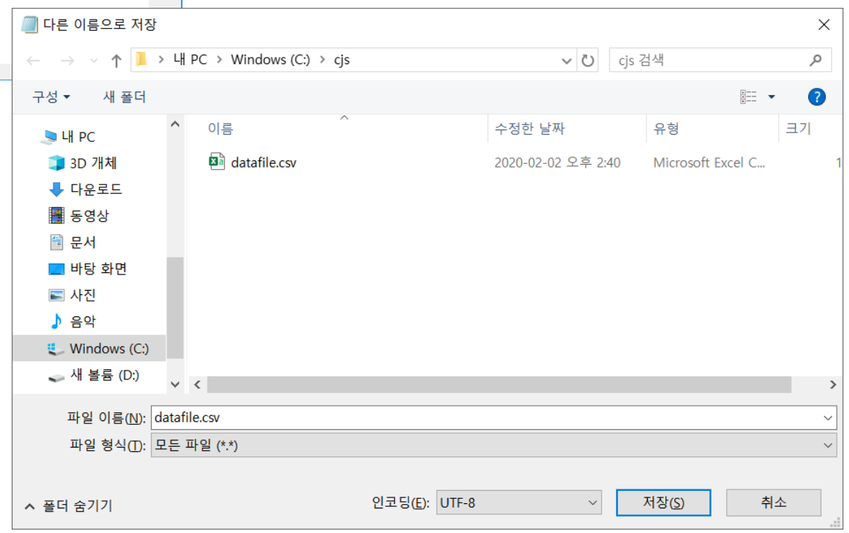
→ 쥬피터노트북이 설치된 같은 폴더 내 C:\사용자\User

Input/Output

■ 넘파이 함수 genfromtxt의 사용 – 텍스트 데이터 불러오기

1. 메모장에서 datafile.csv 파일을 만들고 폴더 C:\cjs에 저장





Input/Output

❷ genfromtxt 함수를 사용하여 데이터 파일을 읽어들인다.

```
arr = np.genfromtxt('C:/cjs/datafile.csv', delimiter=',')
arr
array([[ 1.,   2.,  3.,  4.,   5.],
       [  6.,   7.,  8.,  9.,  10.],
       [ 11., 12., 13., 14., 15.]] )
```

❸ 데이터 3개 열에 이름을 no1, no2, no3로 각각 붙여서 처리 하는 경우

```
arr = np.genfromtxt('C:/cjs/datafile.csv',
                         names=['no1', 'no2', 'no3'], delimiter=',')
arr
array([( 1., 2., 3.), ( 6., 7., 8.), ( 11., 12., 13.)],
       dtype=[('no1', '<f8'), ('no2', '<f8'), ('no3', '<f8')])
arr[0]
( 1., 2., 3.)
arr.dtype
dtype([('no1', '<f8'), ('no2', '<f8'), ('no3', '<f8')])
```

Input/Output

4 names 옵션을 사용하는 경우

```
arr = np.genfromtxt('C:/cjs/datafile.csv', names=True, delimiter=',')
arr
array([( 6., 7., 8., 9., 10.), ( 11., 12., 13., 14., 15.)],
       dtype=[('1', '<f8'), ('2', '<f8'), ('3', '<f8'),
   ('4', '<f8'), ('5', '<f8')])
```

■ 저장 – np.save('C:\cjs\filenametosave.csv', arr, delimiter=',')

■ 원시 2진 파일(raw binary file) 함수 종류

함수의 종류	기능 설명
fromfile(file[, dtype, count, sep])	텍스트나 2진 파일로 된 데이터로부터 배열을 생성
ndarray.tofile(fid[, sep, format])	배열을 텍스트나 2진의 파일로 쓴다.

Input/Output

■ 메모리 매핑(mapping) 파일

❶ memmap 적용을 위한 data 배열 생성

```
data = np.arange(12, dtype='float32')
data.resize((3,4))
data
array([[ 0., 1.,  2.,  3.],
       [ 4., 5.,  6.,  7.],
       [ 8., 9., 10., 11.]], dtype=float32)
```

❷ tempfile 모듈의 mkdtemp() 함수 실행에 의한 임시 디렉토리 생성

→ 생성된 디렉토리에 newfile.dat를 붙인다.

→ 경로 조정을 위해 os.path 모듈을 임포트

```
from tempfile import mkdtemp
import os.path as path
fname = path.join(mkdtemp(), 'newfile.dat')
fname
'C:\\Users\\jchae\\AppData\\Local\\Temp\\
    tmp8hxj7q2j\\newfile.dat'
```

Input/Output

3. data에 매칭하는 dtype와 shape를 저장장치에 이진파일로 저장된 배열에 대한 memmap을 생성

```
fp=np.memmap(fname, dtype='float32', mode='w+', shape=(3,4))
fp
memmap([[ 0., 0., 0., 0.],
        [ 0., 0., 0., 0.],
        [ 0., 0., 0., 0.]], dtype=float32)
```

4. data를 memmap 배열에 쓴다.

→ numpy.memmap의 클래스의 속성인 매핑된 파일에 대한 경로인 filenam의 값과 fname의 절대경로 값이 같은지를 묻는다.

→ memmap객체를 떠나기 전에 저장장치의 메모리 변화를 제거

```
fp[:] = data[:]
fp
memmap([[ 0., 1.,  2.,  3.],
        [ 4., 5.,  6.,  7.],
        [ 8., 9., 10., 11.]], dtype=float32)
fp.filename == path.abspath(fname)
True
del fp
```

Input/Output

5. memmap를 로딩하고 data가 저장되었는지를 확인

```
newfp = np.memmap(fname, dtype='float32', shape=(3,4))
newfp
memmap([[ 0., 1.,  2.,  3.],
        [ 4., 5.,  6.,  7.],
        [ 8., 9., 10., 11.]], dtype=float32)
```

6. 읽기만 가능한 memmap 객체를 생성

```
fpr = np.memmap(fname, dtype='float32',mode='r', shape=(3,4))
fpr.flags.writeable
False
```

7. Copy-on-write memmap 객체를 생성

```
fpc = np.memmap(fname, dtype='float32', mode='c', shape=(3,4))
fpc.flags.writeable
True
```

Input/Output

8. 저장장치에 있는 파일은 읽기 전용

```
fpc
memmap([[ 0., 1.,  2.,  3.],
        [ 4., 5.,  6.,  7.],
        [ 8., 9., 10., 11.]], dtype=float32)
fpc[0,:] = 0
fpc
memmap([[ 0., 0.,  0.,  0.],
        [ 4., 5.,  6.,  7.],
        [ 8., 9., 10., 11.]], dtype=float32)
```

9. 저장장치에 있는 파일은 변동사항이 없다.

```
fpr
memmap([[ 0., 1.,  2.,  3.],
        [ 4., 5.,  6.,  7.],
        [ 8., 9., 10., 11.]], dtype=float32)
```

Input/Output

10 매개변수 옵셋을 적용

```
fpo = np.memmap(fname, dtype='float32', mode='r', offset=16)
fpo
memmap([4., 5., 6., 7., 8., 9., 10., 11.], dtype=float32)
```

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Image Processing

■ 이미지 처리 시 공통적으로 고려되는 사항

→ 이미지의 입력과 출력

→ 불필요한 부분을 다듬는 크로핑(cropping), 뒤집는 플리핑(flipping) 및 회전하기(rotating) 등

→ 잡음 제어(de-noising), 선명히 하기(sharpening) 등의 이미지 필터링(filtering)

→ 다른 객체에 따른 픽셀을 라벨링하여 이미지를 픽셀 단위의 여러 부분으로 분류하기

→ 이미지 분류(classification)

→ 특징 추출(feature extraction)

→ 등록(registration)

■ NumPy와 Matplotlib의 활용

→ C:\아래의 Users 폴더에 있는 파일 python.png를 처리

→ imread함수 내 인수를 ‘C:\\Users\\python.png’ 또는 ‘C:/Users/python.png’라고 표기

→ D폴더 아래의 CJS폴더에 있다면, ‘D:/CJS/python.png’, ‘D:\\CJS\\python.png’

→ 또는 ‘D:\CJS\python.png’라고 입력

Image Processing

```
import numpy as np
import matplotlib.pyplot as plt
arr1 = plt.imread('/Users/python.png')
plt.imshow(arr1)
arr1.shape
(720, 1280, 3)
plt.show()
```


```
def togray(rgb):
...     fil = [0.299, 0.587, 0.144]
...     return np.dot(rgb, fil)
arr2 = togray(arr1)
plt.imshow(arr2, cmap='gray')
plt.show()
```

Image Processing

■ scipy.misc 모듈 – 이미지 처리 기능

함수의 종류	기능 설명
ascent()	8비트 그레이스케일의 512x512인 이미지를 획득
central_diff_weights(Np[, ndiv])	Np-point central derivative에 대한 비중을 반환
derivative(func, x0[, dx, n, args, order])	어떤 지점에서 함수의 n-th derivative를 구한다.
face([gray])	미국너구리 얼굴(raccoon face)의 1024x768인 색상 이미지를 구한다.
electrocardiogram()	1차원 신호에 대한 예제로서 심전도를 로딩한다.

Image Processing

■ scipy로부터 너구리 이미지 얻기

```
import scipy.misc
import matplotlib.pyplot as plt
face = scipy.misc.face()
face.shape
(768, 1024, 3)
face.max(), face.min(), face.mean()
(255, 0, 110.16274388631184)
face.dtype
dtype('uint8')
plt.gray()
plt.imshow(face)
<matplotlib.image.AxesImage at 0xbaa470>
plt.show()
```

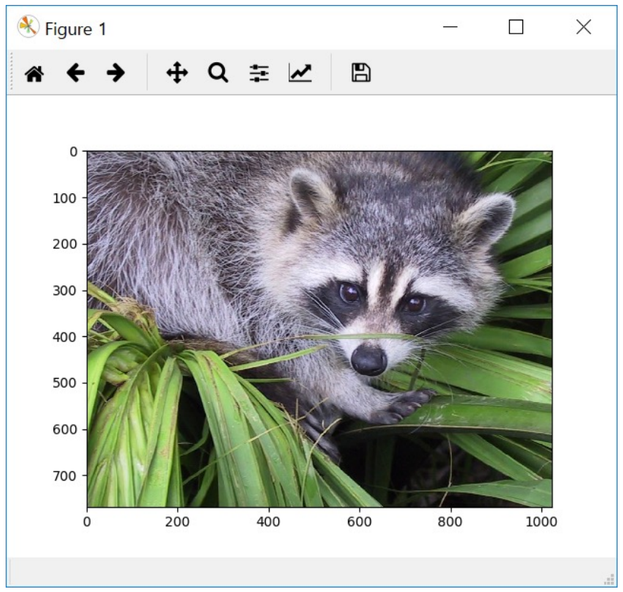


Image Processing

■ 미국너구리 이미지의 원시파일 생성 후 넘파일 배열 생성

```
import imageio
imageio.imwrite('face.png', face)
face1 = imageio.imread('face.png')
type(face1)
imageio.core.util.Image
face1.dtype, face1.shape
(dtype('uint8'), (768, 1024, 3))
face1.tofile('face1.raw')
arr = np.fromfile('face1.raw', dtype=np.uint8)
arr
array([121, 112, 131, ..., 92], dtype=uint8)
arr.shape
(2359296,)
arr.shape = (768, 1024, 3)
```

Image Processing

■ 커다란 데이터의 경우 numpy.memmap를 적용하여 넘파이 배열 arr1

```
arr1=np.memmap('face1.raw',dtype=np.uint8, shape=(768, 1024, 3))
arr1.dtype, arr1.shape
(dtype('uint8'), (768, 1024, 3))
```

■ matplotlib의 함수인 imshow의 적용

```
import scipy.misc
import matplotlib.pyplot as plt
face1 = scipy.misc.face(gray=True)
plt.imshow(face1, cmap=plt.cm.gray)
<matplotlib.image.AxesImage object at 0x00000204BD79DC50>
plt.imshow(face1, cmap=plt.cm.gray, vmin=20, vmax=200)
<matplotlib.image.AxesImage object at 0x00000204BD79DF60>
plt.axis('off')
(-0.5, 1023.5, 767.5, -0.5)
plt.show()
```

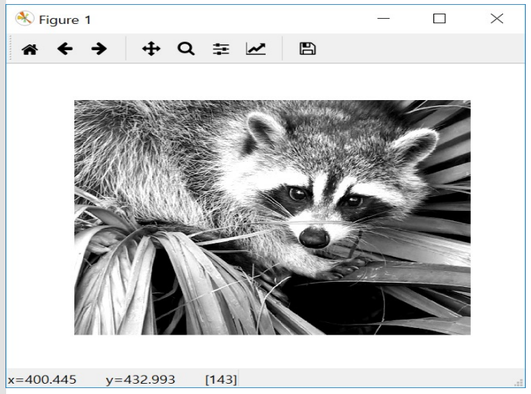


Image Processing

■ 너구리 그림의 일부만을 표시하기

```
import scipy.misc
import matplotlib.pyplot as plt
face1 = scipy.misc.face(gray=True)
plt.imshow(face1[0:500, 400:900], cmap=plt.cm.gray, interpolation='nearest')
<matplotlib.image.AxesImage object at 0x00000204BDAB0CC0>
plt.show()
```

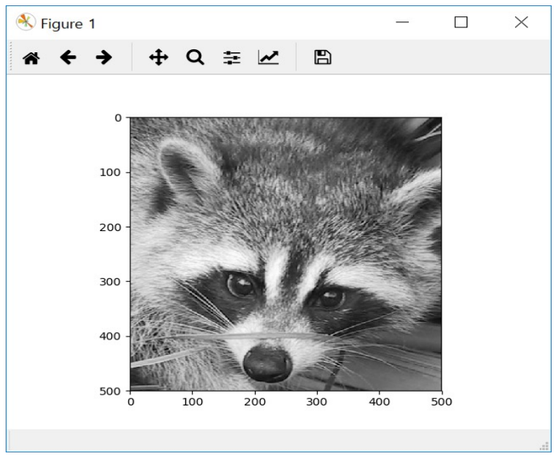


Image Processing

■ scipy로부터 너구리 이미지 얻기

```
import scipy.misc
import matplotlib.pyplot as plt
face = scipy.misc.face()
face.shape
(768, 1024, 3)
face.max(), face.min(), face.mean()
(255, 0, 110.16274388631184)
face.dtype
dtype('uint8')
plt.gray()
plt.imshow(face)
<matplotlib.image.AxesImage at 0xbaa470>
plt.show()
```

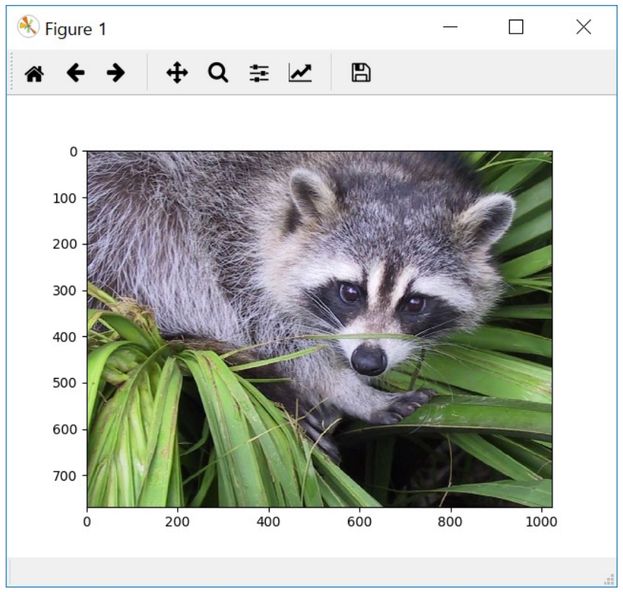
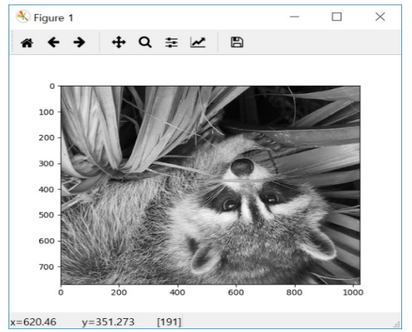
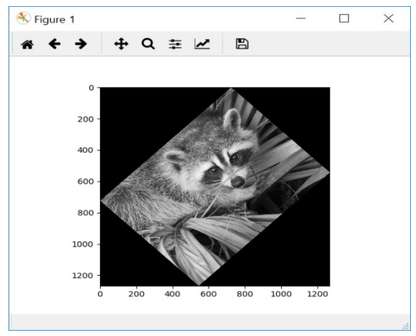


Image Processing

■ 너구리 그림을 뒤집고 회전하는 이미지를 구현하기

```
import numpy as np
import scipy.misc
import scipy.ndimage
import matplotlib.pyplot as plt
arr = scipy.misc.face(gray=True)
arr_flipud = np.flipud(arr)
arr_rotate = scipy.ndimage.rotate(arr, 45)
plt.imshow(arr_flipud, cmap=plt.cm.gray)
<matplotlib.image.AxesImage object at 0x00000246EF916A90>
plt.imshow(arr_rotate, cmap=plt.cm.gray)
<matplotlib.image.AxesImage object at 0x000001B9DCBD5B00>
plt.show()
```





Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- sklearn image case
- Conclusion

scikit-learn

- 2007년에 최초로 releas된 패키지

  →Scipy toolkit에서 따옴

  → 머신러닝 학습 전용으로 거의 default로 쓰이는 패키지

  → 2021년 9월 드디어 Version 1.0 완료

scikit-learn만으로 모든 것을?

- 왠만한 건 거진 다 있음

scikit-learn이 딥러닝?

- Scikit-learn은 기계학습 전용임

  →딥러닝은 다른 패키지로 활용하는 것이 적합함

  →Scikit-learn =\= keras, tensorflow

  → 오른쪽 패키지 학습을 위해서는 별도 자료 필요

How to use it

■ 생각보다 간단합니다.

→ 아래의 과정을 따르시면 됩니다.

→ 단 scikit-learn에만 적용

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
X, y = make_classification(n_samples=100, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=1)
clf = MLPClassifier(random_state=1, max_iter=300)
clf.fit(X_train, y_train)
clf.predict_proba(X_test[:1])
clf.predict(X_test[:5, :])
clf.score(X_test, y_test)
```

① 데이터 로드

② 모델 로드

③ 모델 훈련

④ 데이터 샘플 테스트

⑤ 데이터 테스트

⑥ 모델 정확률 계산

scikit-learn example

- Recognizing hand-written digits

  →https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py

https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py

대체로 학습:테스트는 8:2 비율로 한다

Week 4 Outline

- Week 3 Review
- Universal Function
- View/Copy in Array Object Management and Operation
- Broadcasting
- Array Manipulation and Alignment
- Array Operation
- Input/Output
- Image Processing
- Conclusion

Week 4 Key Takeaway

- 넘피내 유니버셜 기능 및 어레이 매니지먼트 학습

  → 뷰/카피 및 브로드캐스팅 포함한 다양한 기능 학습

- 이미지 처리 통한 실제 numpy 연계 실습

  → 너구리 이미지 통한 모듈 불러오기 및 색감 실습

Assignment 1

총 두 개의 .py 파일만 제출해야 하며 다른 파일 제출 허용하지 않음
부분점수 있음

1. 하기와 같은 코드가 존재한다. <Fill in the blank> 부분을 채워서 최종 출력(초록색 박스)와 같이 도출하시요 다만 하기 조건을 따르시오. (2점)

   → <Fill in the blank> 부분은 8줄 이하로 작성

   → 한줄에다 전체 데이터 입력 처리 불가능

```
import numpy as np
a = np.zeros(3, dtype=[('a', 'f2'), ('b', 'i2'), ('c', 'S2')])
<Fill in the blank.>
print(a)
```

첫 번째 자리는 제곱수, 두 번째 자리는 5승인데 마이너스가 있다. 왜 그런지는 맞혀보라. -> 자료형이 int가 아님 바이트 제한이 있음

```
[( 1.,     1, b'a') ( 4.,    32, b'aa') ( 9.,   243, b'aaa') ( 16.,  1024, b'aaaa') ( 25.,  3125, b'aaaaa') ( 36.,  7776, b'aaaaa') ( 49., 16807, b'aaaaa') ( 64., -32768, b'aaaaa') ( 81., -6487, b'aaaaa') (100., -31072, b'aaaaa')]
```

aaaaa도 다섯 글자에서 끝나는 이유가 있다. 마찬가지로 바이트 제한이 있을 것

Assignment 1

2. 하기와 같은 코드가 존재한다. <Fill in the blank> 부분을 채워서 제출하시요. 다만 하기 조건을 따르시오. (2점)

→ <Fill in the blank> 부분은 5줄 이하로 작성 교수님 답안 : 3줄

→ 차원은 가로 10, 세로 10이고 높이는 data_len 나누기 100을 해야 됨

→ 높이 차원축을 기반으로 평균 도출

→ 10 x 10 결과값을 “test.csv” 파일로 저장(엑셀로 오픈 가능해야 함)

→ data_len 길이는 최소 100000이상이어야 함

```
import numpy as np
data_len=int(input(“길이를 입력하시요.:”))
x = np.random.randn(data_len)
<Fill in the blank.>
```

다음 장에서는

- 분류성능의 평가
- 예측성능의 평가