numpy

다차원 행렬을 빠르고 효과적으로 처리하게 도와주는 라이브러리

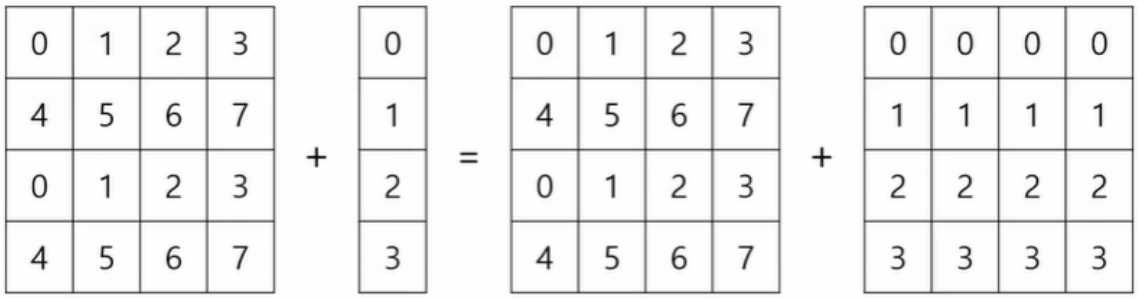

```
axis2 == channel
```

행렬 기본 연산

Numpy의 연산과 함수

서로 다른 형태의 Numpy 연산

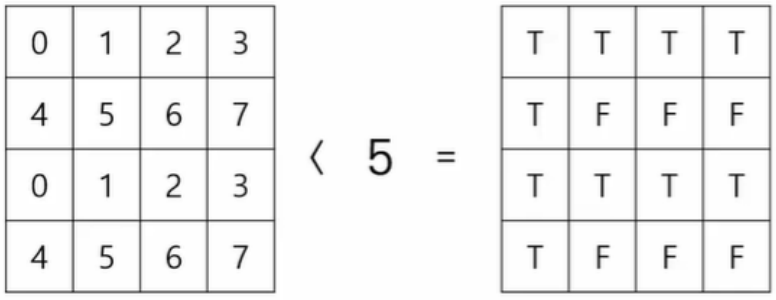
- 브로드캐스트: 형태가 다른 배열을 연산할 수 있도록 배열의 형태를 동적으로 변환



Numpy의 연산과 함수

Numpy의 마스킹 연산

- 마스킹: 각 원소에 대하여 체크합니다.




```
import numpy as np
#기본 행렬 생성 함수
array=np.arange(4)
array
array2=np.zeros((4,4), dtype=float)
array2
array3=np.ones((3,3), dtype=str) #1로 채워진 배열생성
array3
array4=np.full((2,2),7) #특정한 숫자(문자)로 채워진 배열생성
array4

#평균이 0이고 표준편차가 1인 경우 - 표준정규분포
```

```
#배열 합치기
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
arr3 = np.concatenate([arr1,arr2])
arr3.shape

#배열 형태 바꾸기 - 가로(기본)
#[1,2,3,4] -> [1,2
#              3,4]

arr4 = np.array([1,2,3,4])
arr4.reshape((2,2))

#배열 형태 바꾸기 - 세로
arr5 = np.arange(4).reshape(1,4)
arr6 = np.arange(8).reshape(2,4)
arr7 = np.concatenate([arr5,arr6], axis=0)
arr7

#배열 나누기
arr8 = np.arange(8).reshape(2,4)
arr8
left, right = np.split(arr8,[2],axis=1)
left
right

#기본 연산
#기본적으로 사칙연산 지원
#ex) [1,2,3,4] + 2(상수=스칼라) = [3,4,5,6]
#ex) [1,2,3,4] * 2 = [2,4,6,8]
#상수를 곱하면 각각의 데이터에 모두 똑같이 적용

#서로 다른 형태의 배열을 연산할 때는 행 우선으로 수행
#[0,1  + [0,1] = [0,1  + [0,1  = [0,2
# 2,3]            2,3]    0,1] =  2,4]

#브로드캐스트 : 형태가 다른 배열을 연산할 수 있도록 배열의 형태를 동적으로 변환
ar1=np.arange(4).reshape(2,2)
ar2=np.arange(2)
ar1
ar2

#마스킹 : 각 원소에 대하여 체크한다.
masking=arr7 < 4 #4보다 작은 것을 기준으로 bool값 입력
arr7[masking]=100 #True에 해당하는 값만 100을 입력.
arr7

#집계함수 np.max() ,np.min() ,np.sum() ,np.mean()
#집계함수는 특정한 축을 기준으로 출력이 가능
#ex)np.sum(array, axis=0) -> 각 열에 대한 각각의 합계 출력

#행렬의 필요성
#현실 세계에 많은 문제는 행렬을 이용해 해결할 수 있다.
#행렬로 해결이 불가능하다면 해결하기 어려운 문제일 수 있다.
#컴퓨터의 메모리 구조는 행렬 형태로 표현이 가능하다.

#행렬 정렬하기
#기본정렬 - np.sort()
a = np.array([1.5, 0.2, 4.2, 2.5])
b = np.sort(a) #기본값 : 오름차순, 내림차순 -> b[::-1]
print(b)
#[0.2 1.5 2.5 4.2]

#행 or 열을 기준으로 정렬
```

```
sr = np.array([[2, 1, 6],[0, 7, 4],[5, 3, 2]])
np.sort(sr, axis=1) #기본값 : axis=1 (열(좌우)을 기준으로 정렬)
np.sort(sr, axis=0) #axis=0 (행(위아래)을 기준으로 정렬)

#인덱스 값으로 정렬
np.argsort()
a = np.array([1.5, 0.2, 4.2, 2.5])
s = a.argsort() #기본값 : 오름차순, 내림차순 -> s[::-1]
print(s)
#[1 0 3 2]
print(a[s])
#[0.2 1.5 2.5 4.2]
#실제 분석 사용 사례 (kaggle airbnb)
cts += le.inverse_transform(np.argsort(result[i])[::-1])[:5].tolist()

#행렬의 곱 np.dot()
#전제조건 - 연산할 행렬의 열과 다른 행렬의 행의 같아야 함.
#ex) (3,4) * (4,5) = (3(맨 앞의 행),5(맨 뒤의 열))
arr=np.arange(1,13).reshape(3,4)
arr_d=np.arange(1,21).reshape(4,5)
arr_d2=np.arange(1,31).reshape(5,6)
a=np.dot(arr,arr_d)
np.dot(a,arr_d2) #(3,4)*(4,5)*(3,6) = (3,6)
#자세한 행렬의 곱셈 방식은 밑에 사진 참조

#전치행렬 - .T(transpose)
#행렬을 행을 열으로 열을 행으로 뒤바꾸는 행렬을 뜻함.
ar_t=np.arange(1,13).reshape(3,4)
ar_t.T

#행렬의 순회자(itertor) - np.nditer
#1차원의 행렬의 경우는 for문으로 반복이 쉽다.
#2차원 이상의 경우에는 반복문이 복잡해지기 때문에 사용하는 방법
#.finished (bool값) - 행렬의 순회가 끝이 났는지를 bool값으로 처리하는 명령어
#.index, .multi_index - 행렬을 순회할 때 인덱스 값을 기준으로 순회하게 하는 명령어
#1차원(.index), 2차원 이상(.multi_index)
#.iternext() - 다음 순회할 값으로 이동시키는 명령어

#1차원의 경우
i_ar=np.array([1,2,3,4,5])
it_ar=np.nditer(i_ar, flags=['c_index']) #c_index = c언어 스타일의 인덱스 값을 기준으로 순회한다는 의미 (인덱스 값이 0부터 시작)
while not it_ar.finished: #행렬의 모든 값을 순회하기 전까지 무한반복
    idx=it_ar.index       #it_ar이 현재 가리키는 인덱스 위치
    print(i_ar[idx])      #행렬의 인덱스 값으로 적용
    it_ar.iternext()      #다음 인덱스 값으로 이동

#2차원 이상의 경우
i_ar2=np.arange(1,13).reshape(3,4)
it_ar2=np.nditer(i_ar2, flags=['multi_index'])
while not it_ar2.finished:
    idx=it_ar2.multi_index
    print(i_ar2[idx])
    it_ar2.iternext()
※여기서 출력값은 tuple 형태로 출력!
```

행렬의 곱셈 절차

행렬 $A = \begin{bmatrix} a \\ b \end{bmatrix}$와 $B = \begin{bmatrix} c & d \end{bmatrix}$가 있을 때, 이 둘의 곱셈은 다음과 같다.

$$A \times B = \begin{bmatrix} a \\ b \end{bmatrix} \times \begin{bmatrix} c & d \end{bmatrix} = \begin{bmatrix} ac & ad \\ bc & bd \end{bmatrix}$$

$$A^T \times B^T = \begin{bmatrix} a & b \end{bmatrix} \times \begin{bmatrix} c \\ d \end{bmatrix} = ac + bd$$

$$B \times A = \begin{bmatrix} c & d \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = ca + db$$

$$B^T \times A^T = \begin{bmatrix} c \\ d \end{bmatrix} \times \begin{bmatrix} a & b \end{bmatrix} = \begin{bmatrix} ca & cb \\ da & db \end{bmatrix}$$

행렬 $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$와 $B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$가 있을 때, 이 둘의 곱셈은 다음과 같다.

$$A \times B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$