AiDA Lab.
Artificial Intelligence & Data Analysis Laboratory

2021 인공지능 소수전공

5차시: NumPy

2021.07.20 18:30~19:15

Seokhwan Yang

NumPy

- NumPy : Numerical Python
  - 고성능의 과학계산 컴퓨팅과 데이터 분석에 필요한 기본패키지
  - 행렬이나 대규모의 다차원 배열을 쉽게 처리
  - 계산과학(Computational Science) 분야의 복잡한 연산을 지원
  - SciPy, Matplotlib, Pandas 등에 채용되어 더 복잡한 연산을 쉽게 처리 가능하도록 지원

NumPy

- NumPy에서 제공하는 기능
  - 빠르고 메모리를 효율적으로 사용하며 벡터 산술연산과 세련된 브로드캐스팅 기능을 제공하는 다차원 배열 ndarray
  - 반복문을 작성할 필요 없이 전체 데이터 배열에 대해 빠른 연산을 제공하는 표준 수학 함수
  - 배열 데이터를 디스크에 쓰거나 읽을 수 있는 도구와 메모리에 올려진 파일을 사용하는 도구
  - 선형대수, 난수 발생기, 푸리에 변환 기능
  - C, C++, 포트란으로 쓰여진 코드를 통합하는 도구

NumPy

- **C, C++, 포트란으로 쓰여진 코드를 통합하는 도구**
  - 사용하기 편한 C API제공
  - 데이터를 다른 저수준 언어로 쓰여진 외부 라이브러리에 쉽게 전달할 수 있도록 지원
  - 외부 라이브러리에서 반환된 데이터를 파이썬의 NumPy 배열 형태로 불러올 수 있도록 지원
  - 파이썬을 레거시 C/C++/포트란 기반의 코드를 래핑하여 동적이며 쉽게 사용할 수 있는 인터페이스를 만들 수 있는 언어로 만들어 줌

NumPy

- NumPy를 잘 활용하려면
  - NumPy는 자체적으로는 고수준의 데이터 분석 기능을 제공하지 않으므로
  - 먼저 NumPy 배열과 배열기반의 컴퓨팅에 대한 이해가 선행된다면
  - 더 상위 레벨에서 데이터 분석과 같은 기능을 제공하는 Pandas 등의 도구들을 더욱 효율적으로 사용할 수 있음

NumPy

- 대부분의 데이터 분석 애플리케이션에서 중요하게 사용되는 기능
  - 벡터배열 상에서 데이터 개조, 정제, 부분집합, 필터링, 변형, 이종연산의 빠른 수행
  - 정렬, 유일 원소 찾기, 집합연산과 같은 일반적인 배열 처리 알고리즘
  - 통계의 효과적인 표현과 데이터의 수집/요약
  - 이종의 데이터 묶음을 병합하고 엮기 위한 데이터 정렬과 데이터 간의 관계 조작
  - if ~ elif ~ else 포함 반복문 대신 사용가능한 조건절 표현을 할 수 있는 배열 표현
  - 데이터 그룹 전체에 적용할 수 있는 수집, 변형, 함수의 적용과 같은 데이터 처리

NumPy

NumPy는 이런 연산을 위한 기본 라이브러리를 제공함

Pandas와 같은 상위 레벨의 인터페이스를 제공하는 라이브러리와 함께 사용할 때

효과가 극대화됨

ndarray class (n-Dimension Array)

- NumPy의 핵심 기능 중 하나
- 대규모 데이터 집합을 담을 수 있는 빠르고 유연한 자료구조
- 같은 종류의 데이터를 담을 수 있는 포괄적인 다차원 배열

  → ndarray의 모든 원소는 같은 자료형이어야 함
- 배열을 이용한 벡터화 연산
- 메모리에는 연속적으로 저장
- 일반적으로 리스트보다 빠른 것으로 간주됨

Colab

ndarray class (n-Dimension Array)

- **배열 생성 함수**

함수	설명	사용법
array	입력 데이터(리스트, 튜플, 배열 또는 다른 순차형 데이터)를 ndarray로 변환하며 dtype이 명시되지 않은 경우에는 자료형을 추론하여 저장. 기본적으로 입력 데이터는 복사됨	
as array	입력 데이터를 ndarray로 변환하지만 입력 데이터가 이미 ndarray일 경우, 복사가 되지 않음	
arange	내장 range 함수와 유사하지만 리스트 대신 ndarray를 반환	np.arange(1, 100, 5)
ones, ones_like	주어진 dtype과 주어진 모양을 가지는 배열을 생성하고 내용을 모두 1로 초기화. ones_like는 주어진 배열과 동일한 모양과 dtype을 가지는 배열을 새로 생성하여 내용을 모두 1로 초기화	np.ones((3, 6)) np.ones_like(ar)
zeros, zeros_like	ones, ones_like와 같지만 내용을 0으로 채움	np.zeros((3, 6)) np.zeros_like(ar)

ndarray class (n-Dimension Array)

- **배열 생성 함수**

함수	설명	사용법
empty, empty_like	메모리를 할당하여 새로운 배열을 생성하지만 ones나 zeros처럼 값을 초기화하지 않음	np.empty((3, 6))
eye, identity	N x N 크기의 단위행렬을 생성함(좌상단에서 우하단을 잇는 대각선은 1로 채우고 나머지는 0으로 채움)	np.eye(3)
linspace	주어진 범위를 개수만큼 분할	np.linspace(1, 100, 5)
logspace	주어진 범위를 개수만큼 로그로 분할	np.logspace(1, 100, 5)
random.random	0~1사이의 실수로 행렬 생성	np.random.random(10)
random.randint	(시작, 끝) 사이의 임의의 정수의 행렬 생성	np.random.randint(1, 100, (3, 4))
random.randn	-4σ~4σ 사이의 실수로 행렬 생성	np.random.randn(10)
random.sample	0~1 사이의 임의의 수의 행렬 생성	np.random.sample((3, 4))

배열

- **NumPy의 자료형**

종류	Type Code	설명
int8, uint8	i1, u1	부호가 있는 8비트(1바이트) 정수형, 부호 없는 8비트 정수형
int16, uint16	i2, u2	부호가 있는 16비트정수형, 부호 없는 16비트 정수형
int32, uint32	i4, u4	부호가 있는 32비트정수형, 부호 없는 32비트 정수형
int64, uint64	i8, u8	부호가 있는 64비트정수형, 부호 없는 64비트 정수형
float16	f2	반정밀도 부동소수점
float32	f4 또는 f	단정밀도 부동소수점, C언어의 float형과 호환
float64	f8 또는 d	배정밀도 부동소수점, C언어의 double형과 파이썬의 float 객체와 호환
float128	f16 또는 g	확장 정밀도 부동소수점
complex64, complex128, complex25	c8, c16, c32	각각 2개의 32, 64, 128비트 부동소수점 형을 가지는 복소수
bool	?	True, False 값을 저장하는 불리언형
object	O	파이썬 객체형
string_	S	고정길이 문자열형(각 글자는 1바이트). 길이가 10인 문자형의 dtype은 S10
Unicode_	U	고정길이 유니코드형(플랫폼에 따라 글자별 바이트 수가 다름) string_형과 같은 형삭을 사용함(U10)

배열(Arrays)

- NumPy 패키지 임포트

```
import numpy as np
```

  - NumPy의 배열을 사용하기 위해 numpy 패키지 임포트
  - 일반적으로 np라는 이름을 사용함

배열(Arrays)

- 1-D array 생성하기

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

```
arr = np.array(range(1, 6))
arr
```

  - array 명령어를 사용하여 직접 값을 입력할 수 있음

배열(Arrays)

- 2-D array 생성하기

```
ar = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
ar
```

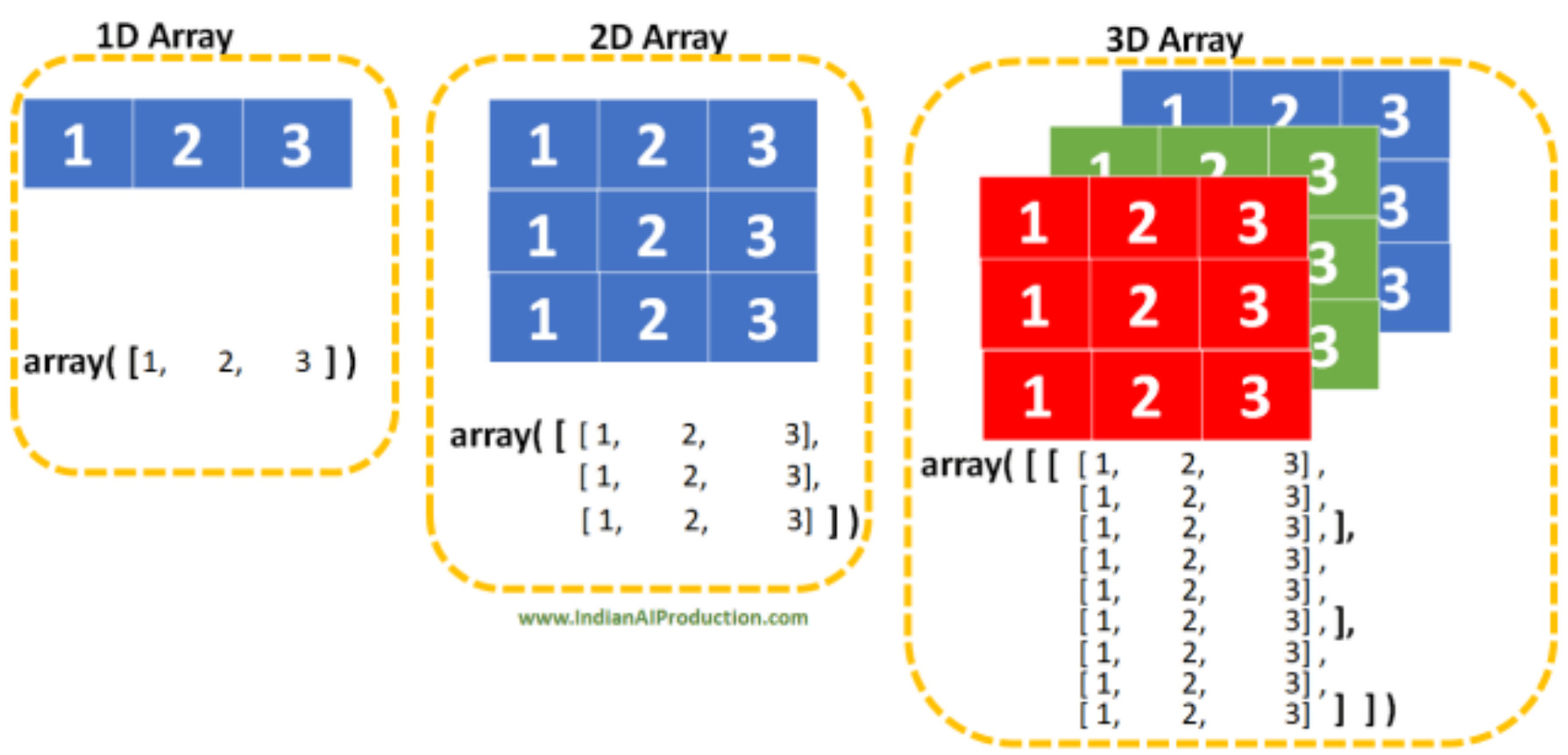
배열(Arrays)

- 3-D array 생성하기

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

배열(Arrays)

- 1-D Array / 2-D Array / 3-D Array



[출처] Python NumPy array - Create NumPy ndarray (multidimensional array) (indianaiproduction.com)

배열(Arrays)

- 명령어

명령어	설명	사용법
shape	행렬(배열)의 모양 확인	np.shape(ar)
ndim	행렬(배열)의 차수(Dimension, 깊이) 확인	np.ndim(ar)
len	행렬(배열)의 길이 확인	len(ar) len(ar[0])
type	행렬(배열)의 형식 확인	type(ar)

배열(Arrays)

- 배열 생성의 예(1)

```
import numpy as np

arrayA = np.zeros((3, 6))
arrayA
```

```
import numpy as np

arrayA = np.ones((3, 6))
arrayA
```

배열(Arrays)

- 배열 생성의 예(2)
  - reshape
    - 배열의 원소의 수는 변하지 않음
    - 차원, 행, 열의 수를 변화시킴

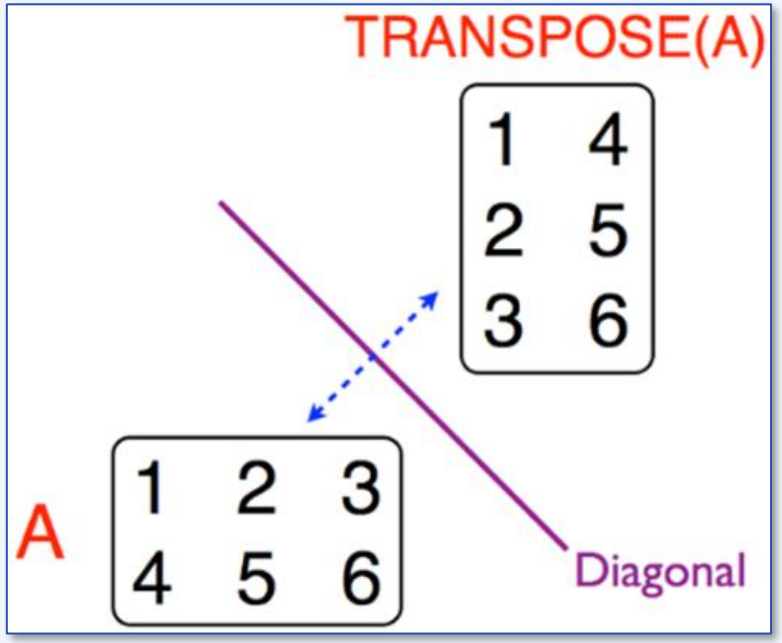
```
import numpy as np

arrayA = np.arange(100, 110).reshape(2, 5)
arrayA
```

```
arr = arrayA.reshape(5, 2)
arr
```

배열(Arrays)

- 변환(Transpose)
  - 배열의 행과 열을 교체
  - ar.T




```
np.arange(10).reshape(2, 5)
```

```
np.arange(10).reshape(2, 5).T
```

배열(Arrays)

- Slicing (1)
  - 1차원 배열: 리스트와 동일

```
ar = array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

ar[1:7]
ar[3:]
ar[:5]
```

배열(Arrays)

- Slicing (2)
  - N차원 배열: ‘,’로 구별하여 연결

```
ar = array([0, 1, 2, 3],
           [4, 5, 6, 7],
           [8, 9, 10, 11]])

ar[0:, 1:]
ar[1:, 1:]
ar[1:2, 1:2]
```

```
ar = np.arange(1, 101).reshape(10, 10)
ar

ar[1:-1, 1:-1]
ar[[1, 3], :]
```

배열(Arrays)

- Concatenation(병합)
  - np.concatenate

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

grid = np.array([[1, 2, 3], [3, 2, 1]])

np.concatenate([grid, grid])

np.concatenate([grid, grid], axis=1)
```

배열(Arrays)

- Concatenation(병합)
  - np.vstack: 수직으로 쌓음
  - np.hstack: 수평으로 병합

```
x = np.array([1, 2, 3])
grid = np.array([[9, 8, 7], [6, 5, 4]])

np.vstack([x, grid])

y = np.array([[99], [99]])

np.hstack([grid, y])
```

배열(Arrays)

- **Splitting (분할)**
  - **np.split / np.hsplit / np.vsplit**

```
grid = np.arrange(16).reshape(4, 4)
grid
```

```
left, right = np.hsplit(grid, [2])
print(left)
print(right)
```

```
upper, lower = np.vsplit(grid, [2])
print(upper)
print(lower)
```

```
left, right = np.hsplit(grid, [3])
print(left)
print(right)
```

```
upper, lower = np.vsplit(grid, [3])
print(upper)
print(lower)
```

배열 연산

$$x = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

$$x + y = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10+0 \\ 11+1 \\ 12+2 \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 14 \end{bmatrix}$$

$$x - y = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10-0 \\ 11-1 \\ 12-2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$$

배열 연산

- Vectorization Operation
  - 배열화 계산
  - 원소 별로 하나씩 계산하는 것이 아니라 배열 단위로 계산
  - Loops 없이 연산하므로 코드의 양 감소
  - 병렬처리 가능(Multi-Core)

배열 연산

- Vectorization Operation
  - 4칙 연산
  - 논리 연산
  - 비교 연산
  - 지수 함수, 로그 함수 지원
  - A[A < 10] = 0
    - 값이 10 이하인 원소를 0으로 치환

```
c = np.array([7, 5, 9, 10, 4, 10, 6, 2]).reshape(2, 4)
d = np.array([1, 3, 4, 9, 14, 7, 6, 4]).reshape(2, 4)
```

```
(c – d) >0
```

```
A[A<10] = 0
```

배열 연산

- 브로드캐스팅(Broadcasting)
  - 행렬끼리 연산할 때 크기가 다른 경우, 이를 알아서 확대해 주는 기능
    - 특별히 작업해 줄 것은 없음
    - 우측의 수행 결과를 확인해 보자

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} =$$

배열 연산

- 브로드캐스팅(Broadcasting)

```
a = np.arange(15).reshape(5, 3)
b = np.arange(5).reshape(5, 1)

c = a + b
```

배열 연산

- 수행 시간 비교
  - 정수 리스트를 2개 만들고 각 원소를 곱하는 연산을 수행해보자
  - %%time을 사용하여 수행시간 확인
    - 셀의 맨 위에 적용. 셀의 평균 실행시간을 표시함

```
%%time

inputA = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
inputB = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

ResultC = inputA * inputB
```

배열 연산

- astype: 자료형 변환

```
a = np.random.random(10)

a = a*100
b = a.astype(int)
```

배열 연산

- dot: 두 행렬의 벡터 내적
  - 행렬의 내적(합성곱)

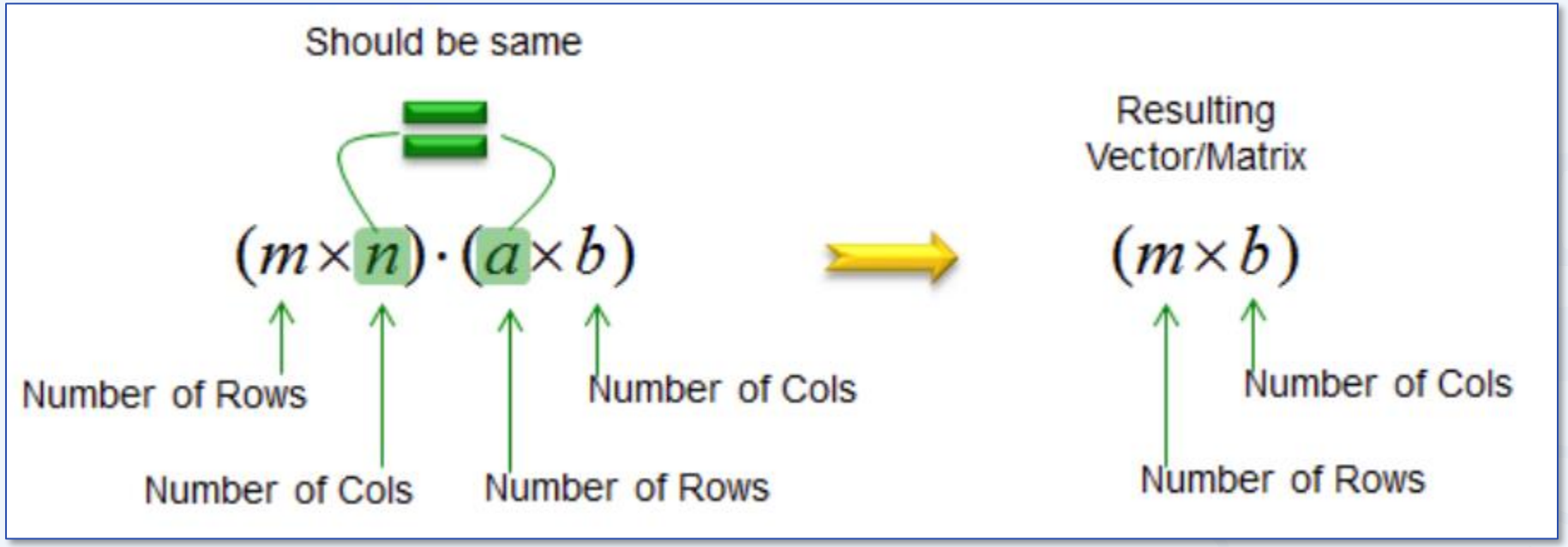
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}) & (a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}) \\ (a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}) & (a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}) \\ (a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}) & (a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}) \\ (a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31}) & (a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32}) \end{bmatrix}$$

$$x \cdot y = < x, y > = x^T y \quad x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad y = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$x^T y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

배열 연산

- dot: 두 행렬의 벡터 내적
  - 행렬의 내적(합성곱)



배열 연산

- dot: 두 행렬의 벡터 내적
  - 행렬과 스칼라 곱
  - 행렬과 행렬 곱

```
ar1 = np.array([1, 2, 3])
ar2 = np.array([1, 2, 3]).T

A = np.dot(ar1, ar2)
```

배열 연산

- 행렬곱은 어디에 활용되는가?
  - 가중치 합(Weighted Sum)
    - 요소들의 동일한 가치로 합산하는 것이 아니라 중요도에 따라 가중치를 곱해준 후 합산

총 가격

$$w_1x_1 + \cdots + w_Nx_N = \sum_{i=1}^{N} w_ix_i$$

각 과일의 개수

각 과일의 가격

$$\begin{bmatrix} w_1 & w_2 & \cdots & w_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

    - 예: 사과, 바나나, 복숭아를 사는데 개수와 가격이 각각 다름
      - 사과 30개, 바나나 500개, 복숭아 70개를 구입한 총액 계산
      - 단가: 사과 1000원, 바나나 170원, 복숭아 1500원

배열 연산

- **행렬곱은 어디에 활용되는가?**
  - **가중치 평균(Weighted Average)**
    - 각각의 가중치를 곱한 합을 전체 가중치의 합으로 나누어 계산
    - 예: 성적 환산 시 학점별로 가중치를 곱하여 평균 계산
      - 학생 A: 국어 87점, 수학 93점, 영어 90점, 도덕 100점
      - 학생 B: 국어 88점, 수학 92점, 영어 93점, 도덕 85점
      - 과목 가중치: 국어 4, 수학 4, 영어 4, 도덕 1
      - 두 학생 중 가중 평균이 더 높은 학생 찾기

배열 연산

- **행렬곱은 어디에 활용되는가?**
  - **유사도 검사 (Similarity Check)**
    - 두 벡터가 비슷할수록 높은 점수가 나옴
    - Cosine 유사도가 많이 사용됨

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

배열 연산

- 유용한 명령어

명령어	설명	사용법
shuffle	아이템의 순서를 뒤섞음	np.random.shuffle(ar)
min	행렬 또는 부분 행렬에서 최소값을 반환	np.min(ar)
max	행렬 또는 부분 행렬에서 최대값을 반환	np.max(ar)
argmin	최소값의 위치 반환	np.argmin(ar)
argmax	최대값의 위치 반환	np.argmax(ar)
sum	행렬의 총합 반환, axis를 기준으로 계산	np.sum(ar, axis=0)
mean	행렬의 평균값 반환, axis를 기준으로 계산	np.mean(ar, axis=1)
median	행렬의 중간값 반환, axis를 기준으로 계산	np.median(ar, axis=0)

배열 연산

- 유용한 명령어

명령어	설명	사용법
var	행렬의 분산(모든 점들에 대한 평균으로부터의 차의 제곱의 합) 반환 작을수록 데이터가 뭉쳐 있음	
std	행렬의 표준편차(분산의 제곱근) 반환 작을수록 데이터가 평균값에 뭉쳐 있음	
random.normal	정규분포자료 생성 - loc: 평균값 - scale: 표준편차 - size: 생성하는 자료의 수	np.random.normal(loc-0.0, scale=1.0, size=None)

배열 연산

- 유용한 명령어
  - sort(정렬)

```
ar1 = np.arange(1, 15)
np.random.shuffle(ar1)

print(ar1)
print(np.sort(ar))
```

```
ar2 = np.arange(1, 16)
np.random.shuffle(ar2)

print(ar2)
print(np.sort(ar2, axis=0))
```

    - 2차원 이상인 경우 axis에 따라 정렬 가능
    - axis=-1이 기본

배열 연산

- 유용한 명령어
  - sqrt(제곱근)

```
ar = np.arange(1, 17)

np.sqrt(ar)
```

  - copy(복사)

```
ar1 = np.arange(1, 17)
print(ar1)

ar2 = np.copy(ar1)
print(ar2)
```

파일 읽기 및 파일 저장

- savetxt
  - 텍스트 파일로 저장

```
np.savetxt('c:/ai/numbers.csv', a, delimiter=',')
```

- loadtxt
  - 텍스트 파일에서 자료 읽기

```
data = np.loadtxt('c:/ai/numbers.csv', delimiter=',')
print(data)
```