주요 개념

스칼라

벡터

행렬

텐서

백터의 내적 np.dot() 백터 끼리 곱의 한 종류

백터의 놈 np.linalg.norm() np.linalg.norm(,1) 정칙화

행렬의 곱 np.dot() 아다마르 곱(요소별 곱) → 일반 사칙연산

전치 a.T

역행렬 A^{-1} np.linalg.inv() 단위행렬(E) np.eye() 정방행렬 행렬식 np.linalg.det()

선형변환 표준기저 벡터그리기 plt.quiver()

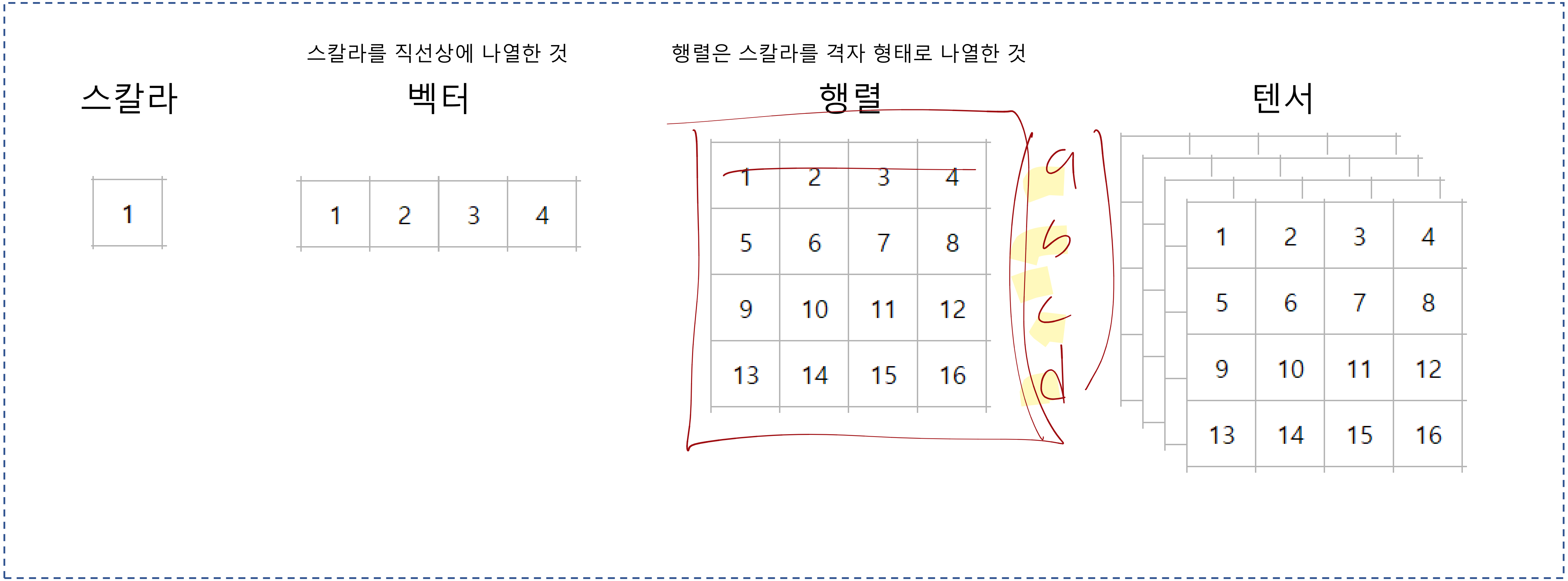
고윳값과 고유백터 np.linalg.eig() 고유방정식 $|A-\lambda E| = \det(A-\lambda E) = 0$

코사인 유사도

스칼라 / 벡터 / 행렬 / 텐서 기본 개념

텐서는 스칼라를 여러 개의 차원으로 나열한 것 스칼라, 벡터, 행렬을 포함한다.

텐서



사용 함수 : array()

벡터의 내적

내적 : 벡터 끼리 곱의 한 종류

두 개 데이터의 상관관계를 구할 때 등에 사용

각 요소끼리 곱한 값을 총합
조건: 두 개의 벡터의 요소 수가 같아야 한다.

```python
# 리스트 4.6_벡터의 내적을 계산한다.
import numpy as np

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

print('-------dot함수-------')
print(np.dot(a,b))
print()
print('-------sum함수-------')
print(sum(a*b))
```

사용 함수 : dot()

Sum()

벡터의 놈

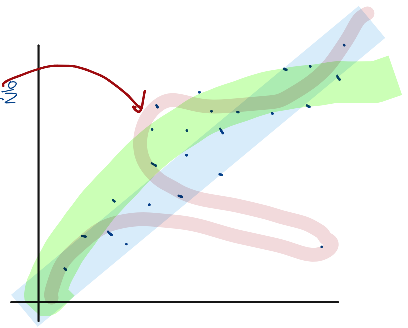
놈 : 벡터의 '크기'를 나타내는 양

L2놈 np.linalg.norm() --> **디폴트가 L2인 함수**
벡터의 각 요소를 제곱합하여 제곱근을 구함

L1놈 np.linalg.norm(,1) --> **1을 줘서 L1놈 구함**
벡터의 각 요소의 절댓값을 더해서 계산

놈은 인공지능에서 정칙화에 쓰인다.
정칙화란 필요 이상으로 네트워크 학습이 진행되는 것을 파라미터로 조절해서 예방하는 것

Over fitting을
방지하는 것



사용 함수 : np.linalg.norm()

linalg는 Linear Algebra라는 선형 대수학의 준말

L2 놈 각 요소 제곱합의 제곱근

$$\|\vec{x}\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 \cdots + x_n^2} = \sqrt{\sum_{k=1}^{n} x_k^2}$$

L1 놈 각 요소 절대값의 합

$$\|\vec{x}\|_1 = |x_1| + |x_2| + |x_3| \cdots + |x_n| = \sum_{k=1}^{n} |x_k|$$

행렬의 곱 vs 아다마르 곱(요소별 곱)

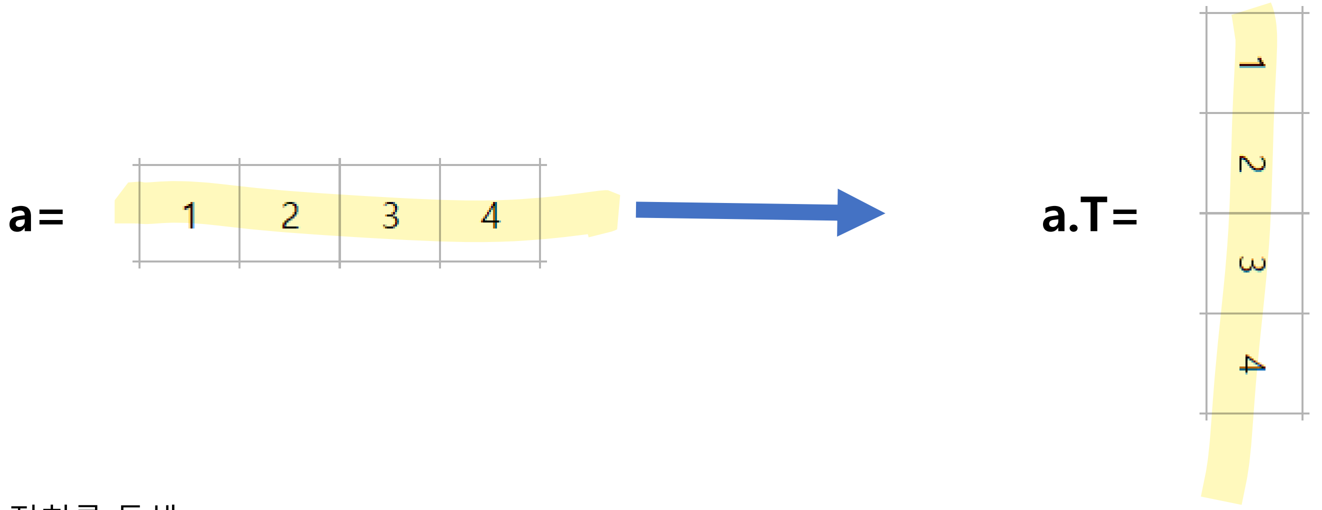
$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \xrightarrow{\text{행렬곱}} \begin{pmatrix} a+3b+5c & 2a+4b+6c \\ d+3e+5f & 2d+4e+6f \end{pmatrix}$$

```
print('-------요소별 곱하기-------')
print(a*b)
print()
print('-------요소별 나누기-------')
print(a/b)
print()
print('-------요소별 더하기-------')
print(a+b)
print()
print('-------요소별 빼기-------')
print(a-b)
```

행렬 곱　사용 함수 : dot()

아다마르 곱　사용 함수 : 일반 연산자

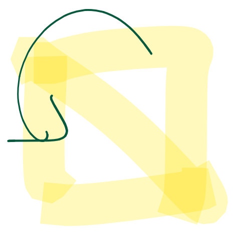
전치 *하는 법: 배열 이름.T*



전치를 통해
앞 행렬의 열수와 뒤 행렬의 행수를 일치시켜
곱을 가능하게 할 수도 있다.

```python
# 리스트 4.14_ 행렬을 전치한다.
import numpy as np

a = np.array([[1,2,3],
              [4,5,6]]) # 2x3의 행렬
print(a)
print()
print(a.T)  # 원본을 바꾸진 않는다.
print()
print(a)
```

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

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

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



단위 행렬(E)　　　사용 함수 : eye()

정수에서 1처럼 곱했을 때 자기 자신이 나오도록 하는 행렬

같은 크기의 행렬에 곱해도 곱하는 대상의 행렬을 변화시키지 않는 성질이 있다.
행과 열 수가 같은 정방행렬의 경우에만 가능하다.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & & \\ & & \ddots & \\ 0 & & & \\ 0 & & & 1 \end{pmatrix}$$

역행렬과 행렬식

A라는 임의의 정방행렬과 곱했을 때 **E**(단위행렬)이 나오도록 하는 행렬 → **사용 함수 : linalg.inv()**

조건: **행렬식=0**이 아니어야 한다. → **사용 함수 : np.linalg.det()**

즉. np.linalg.det()의 값이 0이 아니어야 한다.

$A \times A^{-1} = E$

$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

$|A| = det(A) = ad - bc = 0 \rightarrow$ 역행렬 X

$\neq 0 \rightarrow$ 역행렬 O

역행렬은 인공지능에서 변수끼리의 상관관계를 알아보는 회귀분석에 활용된다.

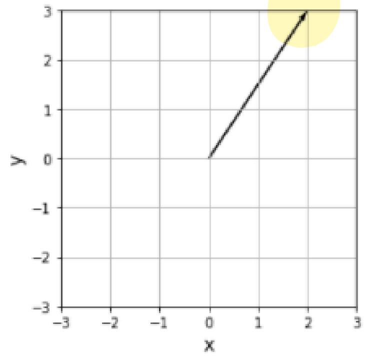
inv는 inverse의 약자

벡터 그리기   사용 함수 : quiver()

quiver(시작점의 x좌표, 끝점의 y좌표, 화살표의 x성분, 화살표의 y성분, angles=화살표의 각도의 결정방법, scale_units=스케일의 단위, scale=스케일, color=화살표의 색)

```python
import numpy
import matplotlib.pyplot as plt

# 화살표를 그리는 함수 만들기
def arrow(start, size, color) :
    plt.quiver(start[0], start[1], size[0], size[1],
               angles="xy", scale_units="xy", scale=1, color=color)


# 화살표의 시작점
s = np.array([0,0]) # 원점

# 벡터
a = np.array([2,3])

arrow(s, a, color='black')

# 그래프 표시
plt.xlim([-3,3])
plt.ylim([-3,3])
plt.xlabel("x", size=14)
plt.ylabel("y", size=14)
plt.grid()
plt.gca().set_aspect("equal")
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



선형변환

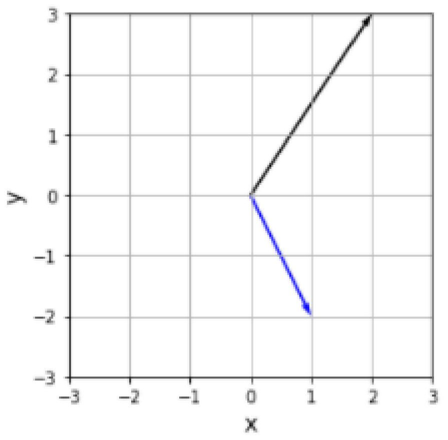
벡터에서 벡터로의 변환을 선형변환이라고 한다.

인공지능에서 뉴럴 네트워크로 정보를 전파시키는데 선형변환을 사용

A가 정방 행렬이 아니면 선형변환에 의해 벡터의 요소수가 변하게 된다.(p154)

```
변환 전 벡터: [2 3]
변환 후 벡터: [ 1 -2]
```

Out[6]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```




```python
import numpy
import matplotlib.pyplot as plt


# 변환 전 벡터
a = np.array([2,3])

A = np.array([[2,-1],[2,-2]])

# 선형변환
b = np.dot(A, a)

print('변환 전 벡터:', a)
print('변환 후 벡터:', b)

# 화살표를 그리는 함수 만들기
def arrow(start, size, color) :
    plt.quiver(start[0], start[1], size[0], size[1],
               angles="xy", scale_units="xy", scale=1, color=color)

# 화살표의 시작점
s = np.array([0,0])  # 원점

arrow(s, a, color='black')
arrow(s, b, color='blue')

# 그래프 표시
plt.xlim([-3,3])
plt.ylim([-3,3])
plt.xlabel("x", size=14)
plt.ylabel("y", size=14)
plt.grid()
plt.gca().set_aspect("equal")
plt.show
```

$\begin{pmatrix} 2 \\ 3 \end{pmatrix}$

$\begin{pmatrix} 2 & -1 \\ 2 & -2 \end{pmatrix}\begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$

표준기저

벡터는 표준기저와 상수의 곱의 합으로 표현할 수 있다.

$$\vec{a} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} = 2\underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\vec{e}} + 3\underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{\vec{e}} = 2\boxed{\vec{e}_x} + 3\boxed{\vec{e}_y}$$

$$\begin{pmatrix} 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 3 \end{pmatrix} \rightarrow \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

고윳값과 고유벡터의 계산_ 고유방정식 사용함수: linalg.eig()

인공지능에서 데이터를 요약하는 주성분 분석이라는 기법에 이용된다.

PCA : Principle Component Analysis)

전처리 단계에서

정의: $A\vec{x} = \lambda\vec{x}$

고윳값 (or 고유치)

고유벡터 (0이 아닌 벡터)

$A\vec{x} - \lambda\vec{x} = 0$

$(A - \lambda E)\vec{x} = 0$

$\vec{x} = (A - \lambda E)^{-1} \times 0$

$= 0$ → $\vec{x}$는 0이 아니라는 조건에 위배

$\therefore$ $A - \lambda E$는 역행렬이 존재할 수 없다.

||는 행렬식을 표시하는기호 → $|A - \lambda E| = \det(A - \lambda E) = 0$ ☆

고유방정식

$A = \begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix}$ 의 고유치 고유 벡터

-1, 4

$A\vec{x} = \lambda\vec{x}$

$(A - \lambda E)\vec{x} = 0$

$A - \lambda E = 0$

$\begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} = \begin{pmatrix} 3-\lambda & 2 \\ 2 & -\lambda \end{pmatrix}$

$-3\lambda + \lambda^2 - 4 = 0$

$\lambda^2 - 3\lambda - 4 = 0$

$(\lambda - 4)(\lambda + 1) = 0$

$-1 \quad 4$

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

$$(A - \lambda E)\vec{x} = 0$$

$$\left(\begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix} - \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \right) \begin{pmatrix} x \\ y \end{pmatrix} = 0$$

$$\begin{pmatrix} 4 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = 0$$

$$4x + 2y = 0 \qquad (1 \quad -2)$$

$$2x + y = 0 \qquad 2x = -y$$

고윳값과 고유벡터의 계산_ 고유방정식　　사용함수: linalg.eig()

다차원을 2차원이나 3차원으로 압축.

인공지능에서 데이터를 요약하는 주성분 분석이라는 기법에 이용된다.

$$\frac{\text{고윳값의 총합}}{\text{고윳값(고유벡터에 대응하는)}} \rightarrow \text{척도가 된다}$$

```python
# 리스트 4.28 linalg.eig() 함수를 사용해 고윳값 고유벡터를 구한다.
import numpy as np

a = np.array([[1,2],
              [3,4]]) # 2x3의 행렬

ev = np.linalg.eig(a)

# 0번 인덱스 요소는 고윳값
print(ev[0])

print()

# 1번 인덱스 요소는 고유벡터
print(ev[1])
```

```
[-0.37228132  5.37228132]

[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

각 열이 고유벡터를 나타낸다.

각 고유벡터는 L2놈이 1이 된다.　고유벡터는 L2놈이 1인 벡터를 단위 벡터라고 한다.

코사인 유사도

사용함수: np.dot() / np.linalg.norm()

벡터끼리의 방향의 가까운 정도.
2개 벡터의 방향이 얼마나 일치하고 있는지를 나타내는 지표

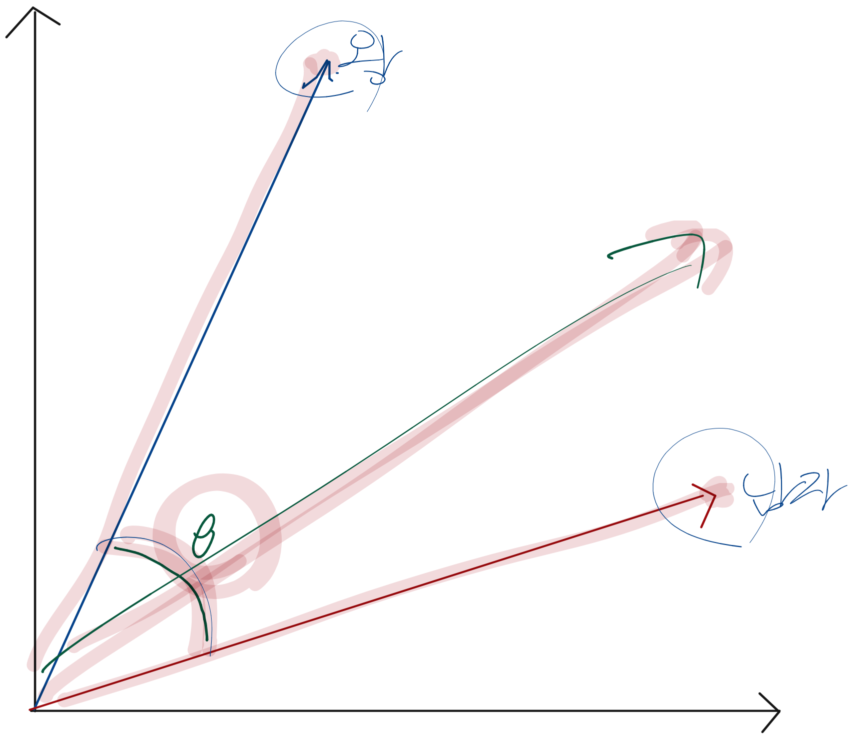
Cosθ의 값은
θ = 0 -> cosθ = 1 (cosθ의 최댓값) -> 유사도가 동일

θ ↑ -> cosθ의 값은 작아진다. -> 유사도가 작아진다.

cosθ = -1 (cosθ의 최솟값) -> 반대

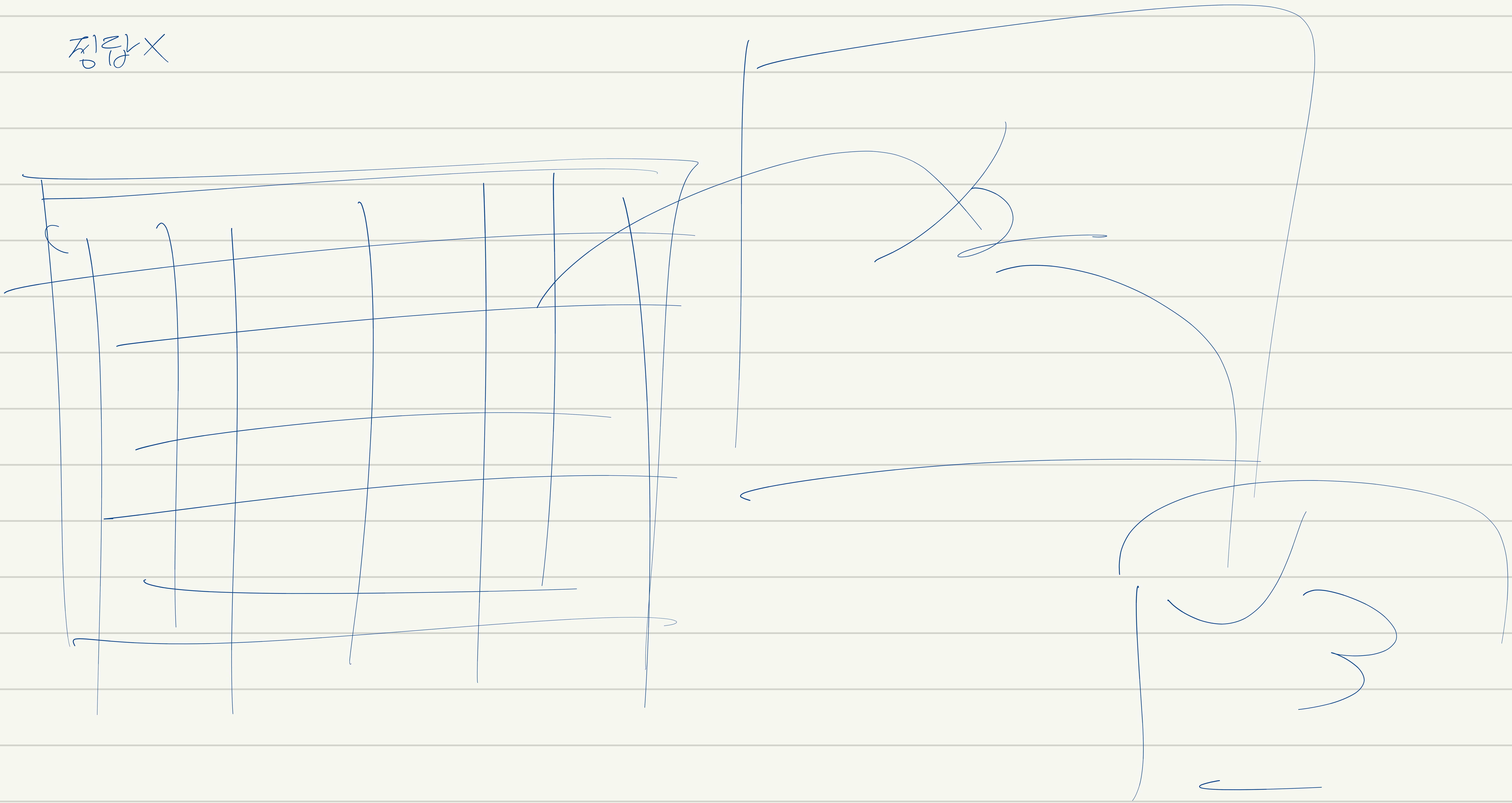
$$\boxed{\cos\theta} = \frac{\text{벡터1} \times \text{벡터2}}{(\text{벡터1의 놈}) \times (\text{벡터2의 놈})}$$

$$= \frac{\text{np.dot(벡터1, 벡터2)}}{\text{np.linalg.norm(벡터1)} \times \text{np.linalg.norm(벡터2)}}$$



지도학습 정답 O

비지도학습 정답 X



강화학습 : ex) 컴퓨터에게 슈퍼마리오를 깨라고 하는

유전 알고리즘 (원리 : 가장 잘 한 걸 pick 해서 학습)