Vector

Vectorization

Vector

Vector, Vectorization

- Scalar: 숫자 하나 (실수)

- **Vector: 숫자의 한 줄 배열 (1D array)**

  크기와 방향을 가짐

- Matrix: 숫자의 사각형 형태 배열 (2D array)

  (직사각형 그리드)

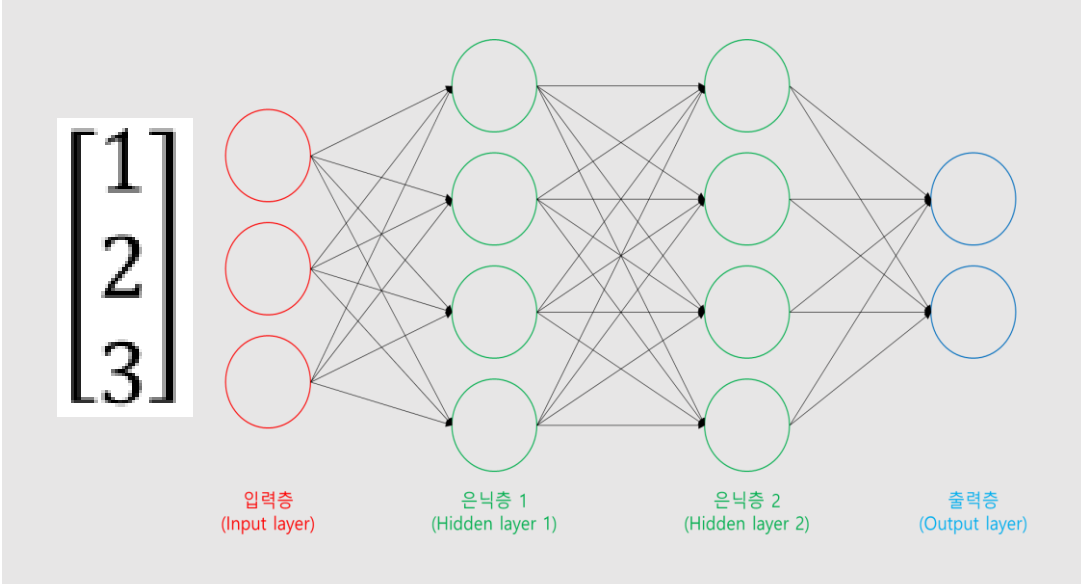
Vector, Vectorization

벡터 (vector)

- 정의
  - ✓ 크기와 방향을 가진 물리량
  - ✓ 벡터 공간(Vector Space)을 이루는 단위 원소
  - ✓ 여러 개의 숫자를 한줄로 배열한 것

- 종벡터, **횡벡터**
  - $a = [1 \quad 2 \quad 3]$, $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

- 원소 (element): 벡터를 구성하고 있는 각 숫자



Vector, Vectorization

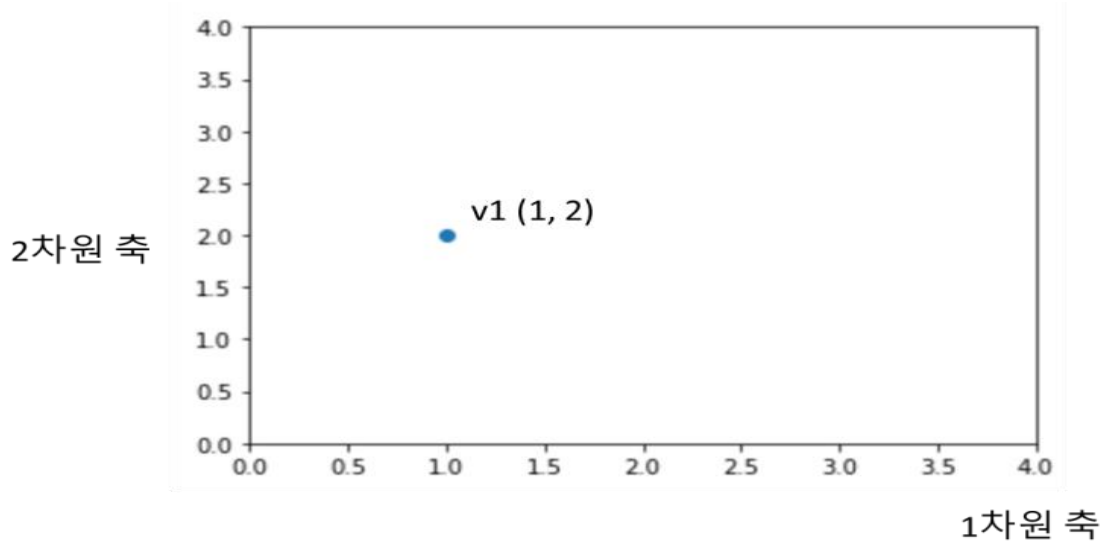
벡터 (vector)

벡터의 기하학적 의미 (공간에서의 의미)

- 하나의 vector는 N 차원 공간 상의 위치를 가짐 (공간 內 point)
- 벡터의 **차원**: 벡터에 포함된 **원소의 수**

  N 차원 벡터 = N개의 원소 수
- ex, 2차원 공간의 벡터 (v1 = [1 2], (1,2))

  = 2차원 공간의 한 점



Vector, Vectorization

벡터 (vector)

벡터의 기하학적 의미 (공간에서의 의미)

- 차원 공간에서의 vector의 위치는 vector의 원소값에 의해 결정
- 벡터 원소 값은 해당 벡터의 **고유한 특성**을 의미

  = **벡터의 공간상에서의 위치는 해당 벡터의 고유한 특성을 반영**

- 차원 공간에서의 vector의 위치 정보를 사용해 **벡터 간의 유사도** 계산 가능

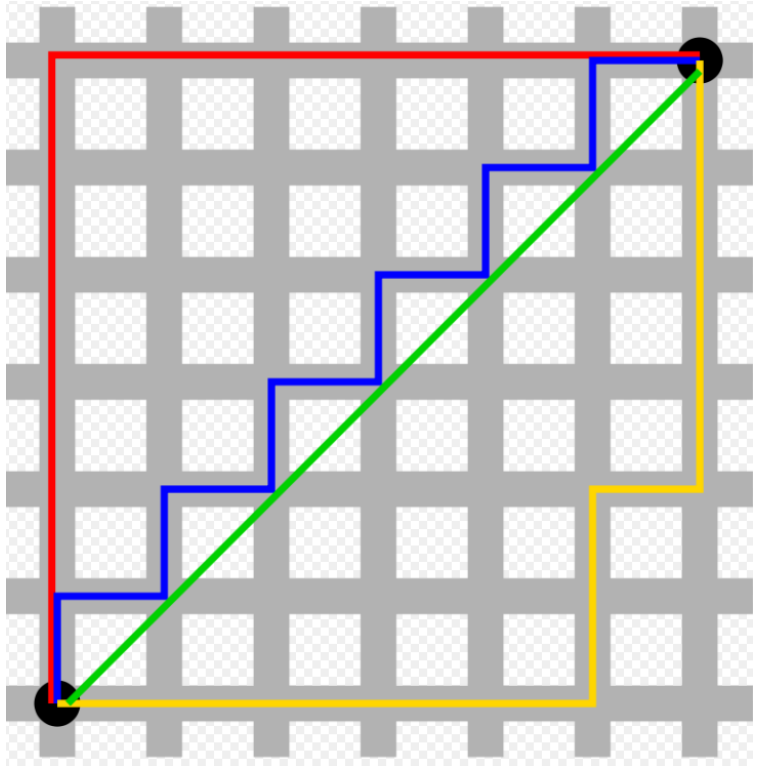
  비슷한 위치 = 높은 유사도 = 유사한 특성
- 위치가 비슷한 정도 → 거리로 계산

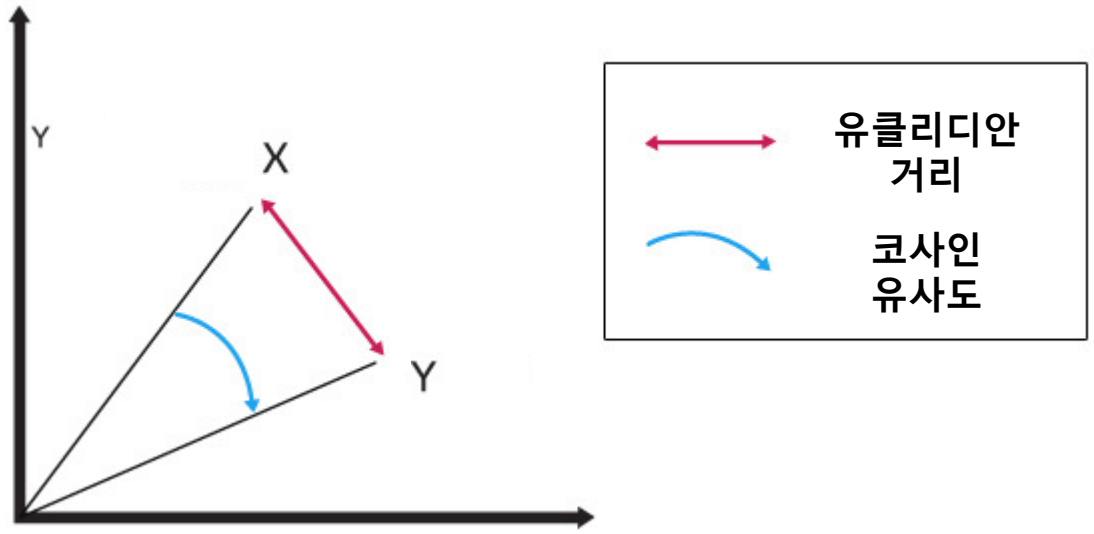
  ✓ 유클리디안 거리, 코사인 유사도 등 사용

Vector, Vectorization

벡터 (vector) - 거리

유클리디안 거리

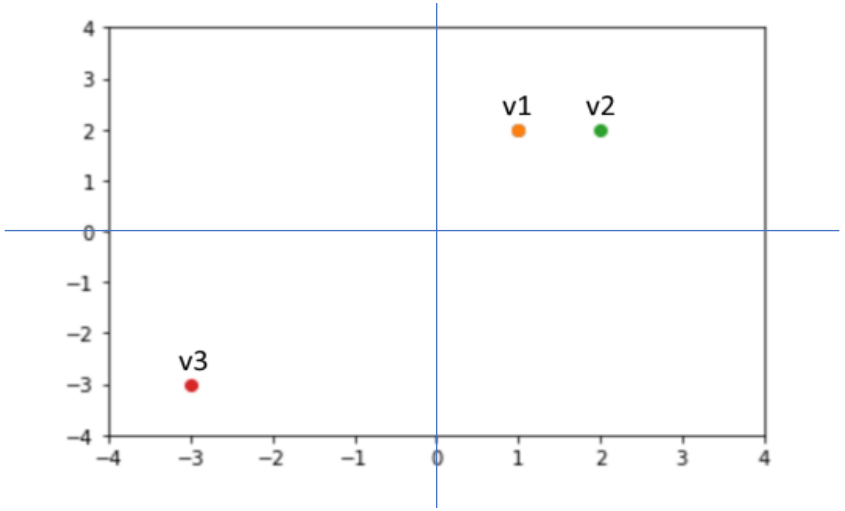




Vector, Vectorization

벡터 (vector) - 거리

- 벡터 거리 = 벡터간의 유사성
  가까운 거리 = 가까운 유사도
- ex: v1 = (1, 2),
  v2 = (2, 2),
  v3 = (-3, -3)



Vector, Vectorization

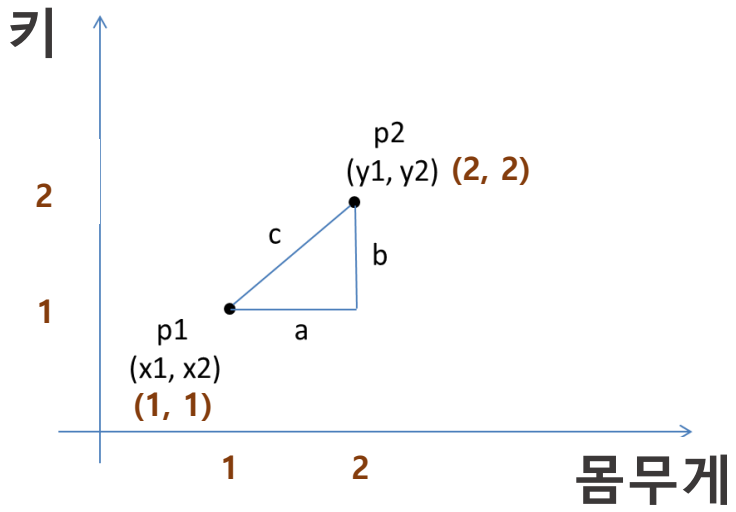
벡터 (vector) - 거리

거리 계산

- **유클리디안 거리**
  - ex, p1 = (x1, x2), p2 = (y1, y2)
  - $\overline{p_1p_2} = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2}$

    p1 = (1, 1), p2 = (2,2)

    $\overline{p_1p_2} = \sqrt{(2-1)^2 + (2-1)^2} = 1.4142$



p1과 원점 사이의 거리

$|v_1|$ = np.linalg.norm(v1) <-- 벡터의 길이(또는 크기를 계산) $\sqrt{2}$

norm? -> **벡터의 길이**(혹은 크기)를 측정하는 방법

cf) $\overline{v_1v_2}$ = np.linalg.norm(v2-v1)

Vector, Vectorization

벡터 (vector) - 거리

거리 계산

- 유클리디안 거리

Q. p1 = (1, 2), p2 = (2,2)

$\overline{p_1 p_2}$?

$|v_1|$?

Vector, Vectorization

벡터 - norm ('표준'이나 '규범'을 의미하는 라틴어 'norma'에서 유래)

놈 : 벡터의 크기(magnitude) 또는 길이(length)를 측정하는 방법

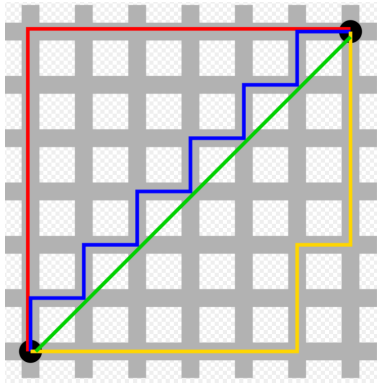
벡터를 구성하는 성분의 값에 기반하여 그 벡터가 얼마나 '큰지'를 나타내는 수치적인 방법

벡터가 원점으로부터 얼마나 떨어져 있는지를 측정하는 수단

다양한 종류의 노름이 있으며 각각은 벡터 공간에서의 벡터 크기를 다른 방식으로 정의

- **L1 norm**

벡터의 각 성분의 절대값의 합으로 계산, "**맨해튼 거리**"
두 점 사이의 거리를 격자와 같은 도시에서 직각으로만 이동할 때의 거리로 생각



- **L2 norm**

유클리드 norm
벡터 성분의 제곱합의 제곱근으로 계산.
이는 기하학적으로 두 점 사이의 '직선 거리'와 동일

두 벡터 사이의 거리
두 벡터 간의 차이(즉, 한 벡터에서 다른 벡터를 뺀 결과)에 대한 norm을 계산

Vector, Vectorization

벡터 – L1norm, L2norm

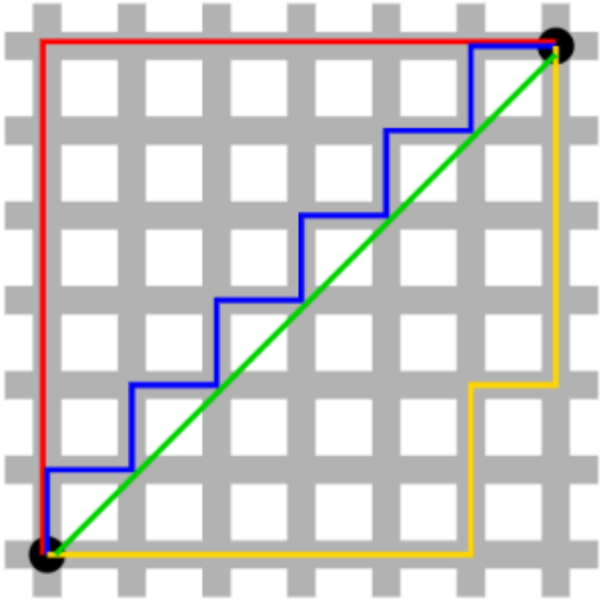
L1norm

$$||x||_1 = \sum_i |x_i| = |x_1| + |x_2| + \ldots + |x_i|$$

L2norm

$$||x||_2 = \sqrt{\left(\sum_i x_i^2\right)} = \sqrt{x_1^2 + x_2^2 + \ldots + x_i^2}$$

L1 norm(red, blue, yellow), L2 norm(green)



Vector, Vectorization

벡터 (vector)

데이터 內의 벡터

관측치	Age	Experience
1	30	1
2	33	2
3	55	25

각 관측치를 벡터로 표현

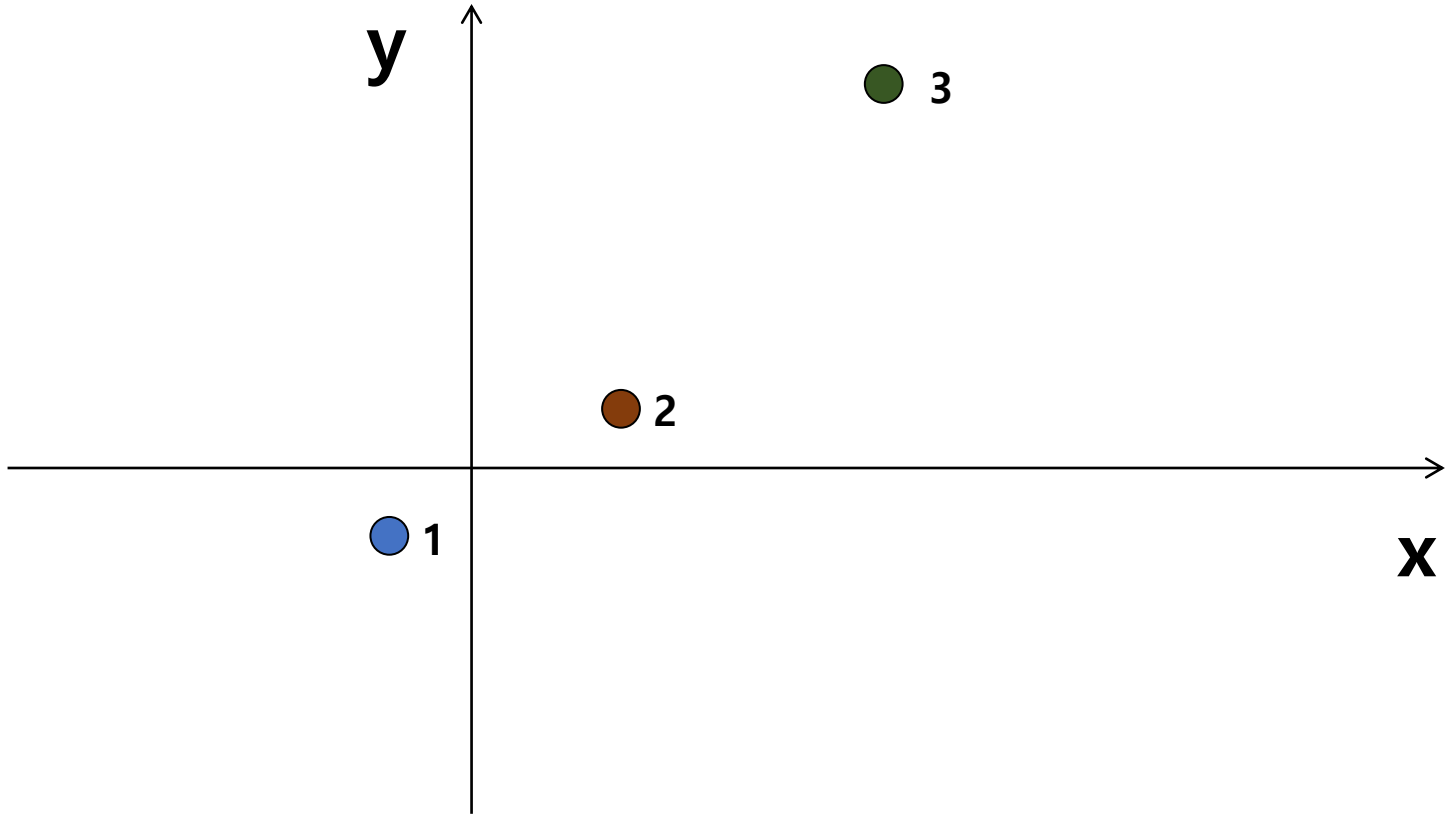
- **데이터의 변수(특성) 정보를 관측치 하나의 벡터로 표현**
  - 1 = (30,1), 2 = (33,2), 3 = (55,25)
  - 독립변수 2개 -> 2차원 벡터
    - ✓ 벡터의 첫번째 원소 값 = 첫번째 변수(Age) 값,
    - ✓ 벡터의 두번째 원소 값 = 두번째 변수(Experience) 값

Vector, Vectorization

벡터 (vector) – 유사도

유사도 = 벡터간 거리

3개의 데이터 포인트(3개의 벡터), 가장 가까운 벡터는?

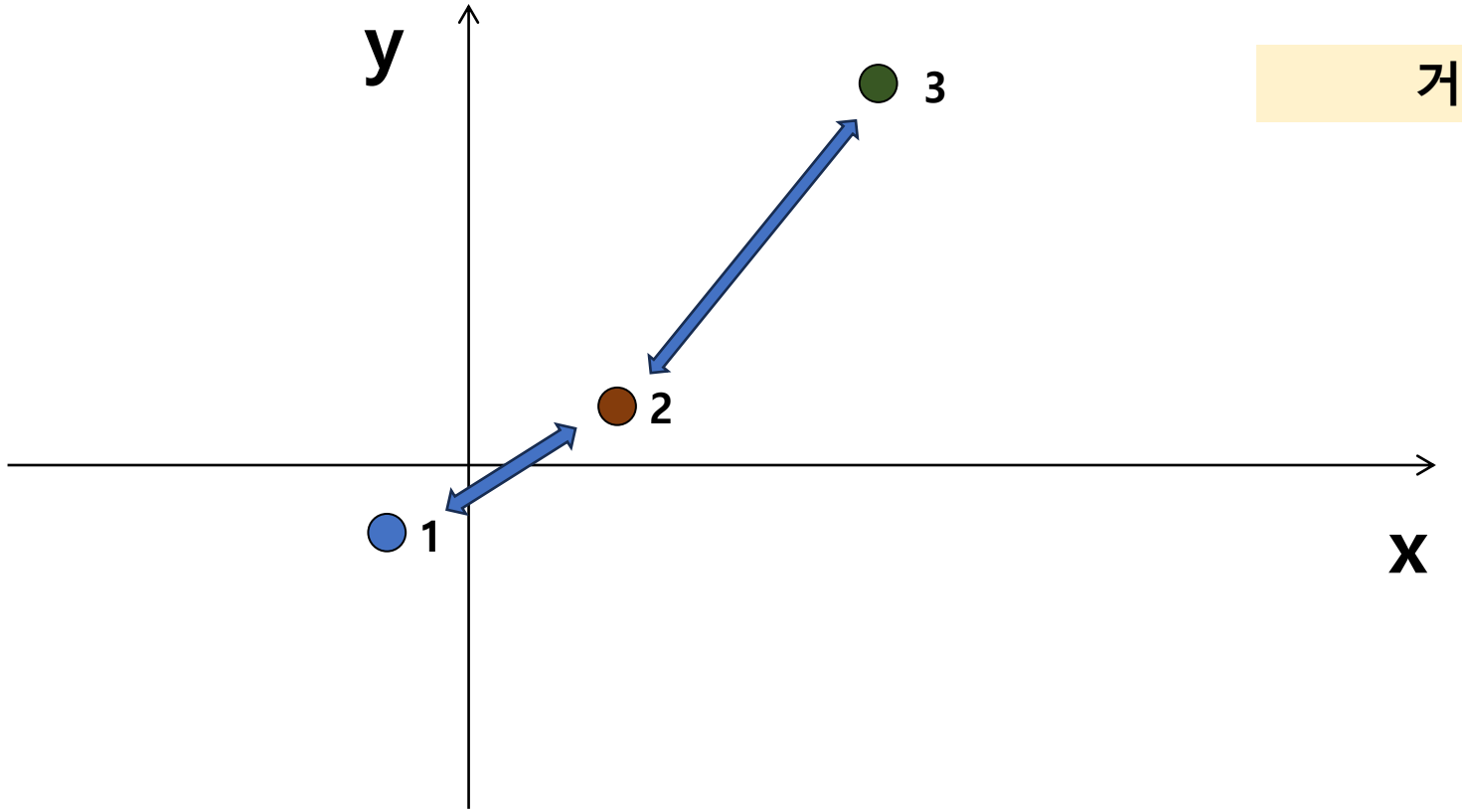


Vector, Vectorization

벡터 (vector) – 유사도

유사도 = 벡터간 거리

3개의 데이터 포인트(3개의 벡터), 가장 가까운 벡터는?



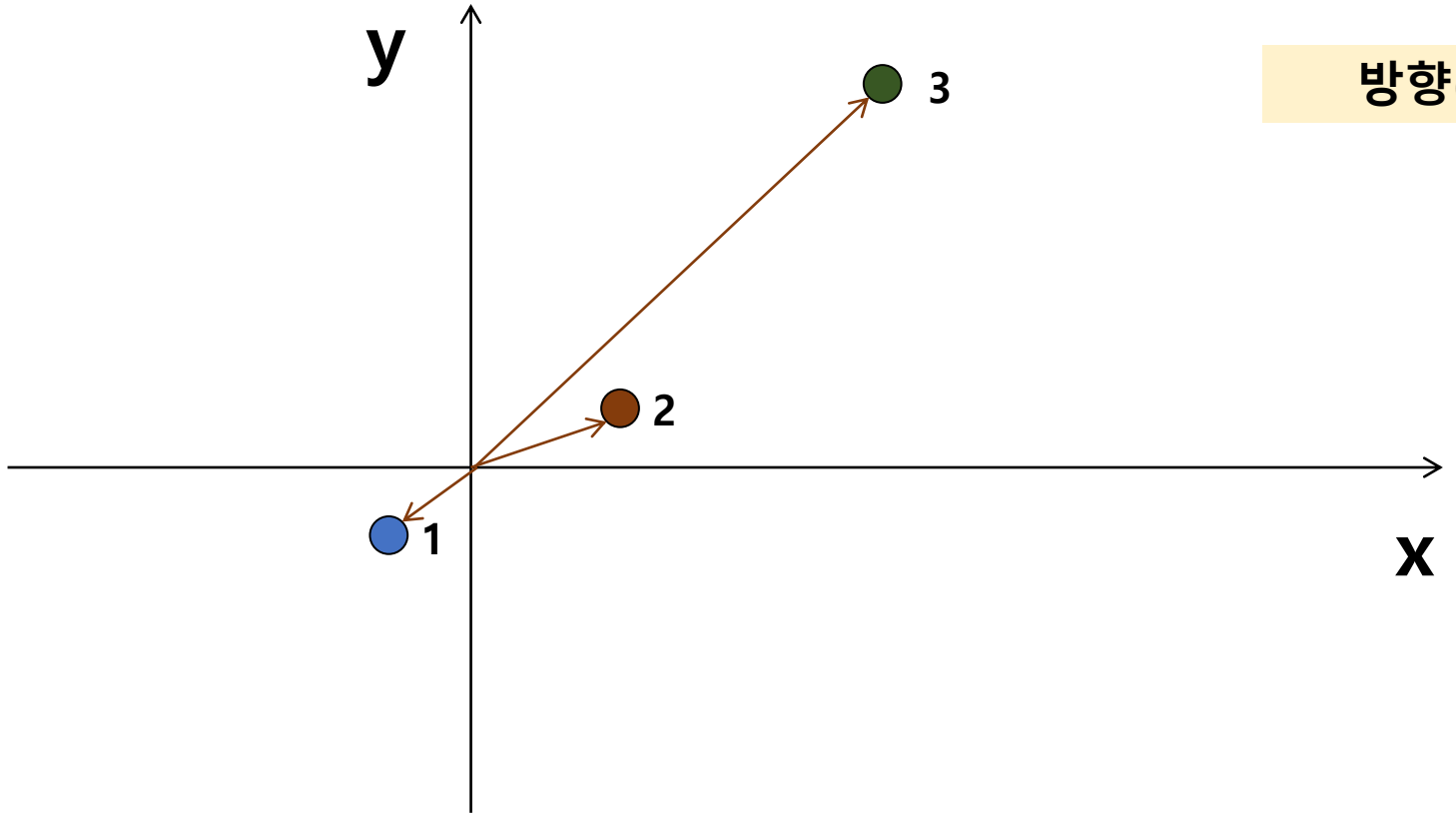
거리

Vector, Vectorization

벡터 (vector) – 유사도

유사도 = 벡터간 거리

3개의 데이터 포인트(3개의 벡터), 가장 가까운 벡터는?



방향(각도)

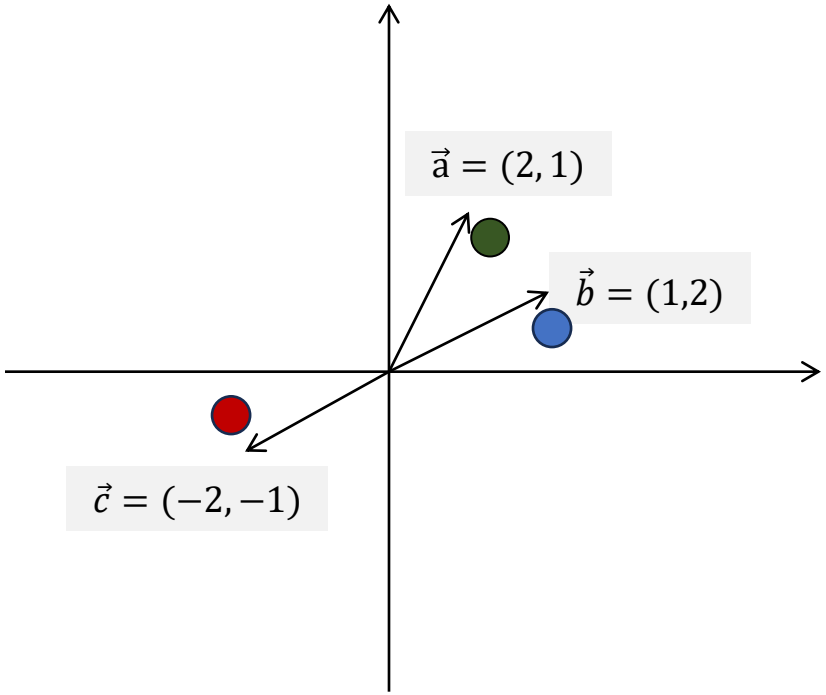
Vector, Vectorization

벡터 (vector) – 유사도 1 - 방향성 기준

유사도 = 벡터간 거리

- 벡터의 방향:
  : 원점을 기준으로 함 (원점에서의 방향)
  : 원점에서 벡터의 끝점까지 연장되는 선(길이와 무관)
  : 벡터의 위치는 벡터의 원소의 값에 의해서 결정
  -> 벡터의 고유한 특성을 반영

- **방향이 비슷함 -> 유사도가 높음**



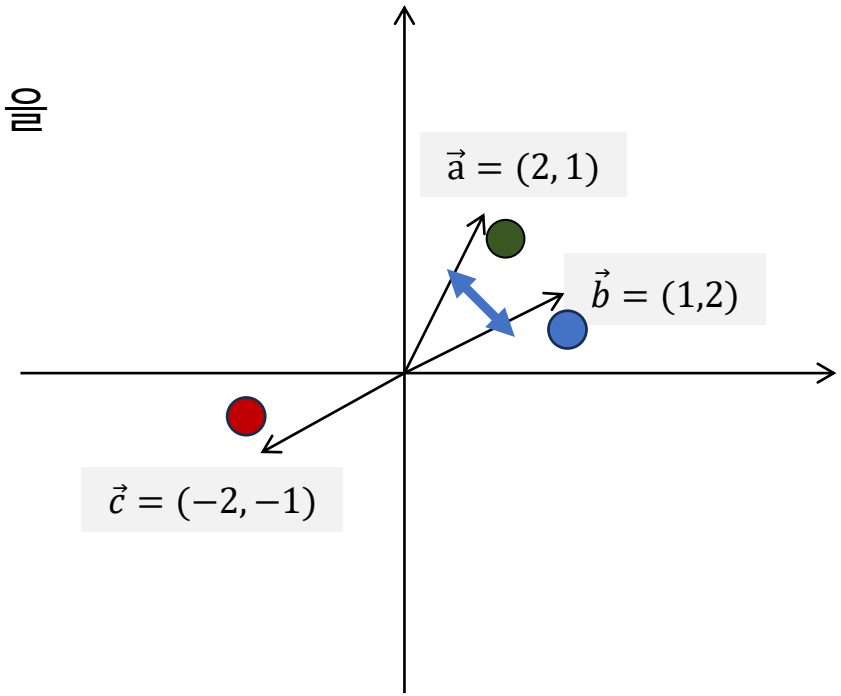
Vector, Vectorization

벡터 (vector) – 유사도 1 - 방향성 기준

유사도 = 벡터간 거리

방향의 유사도

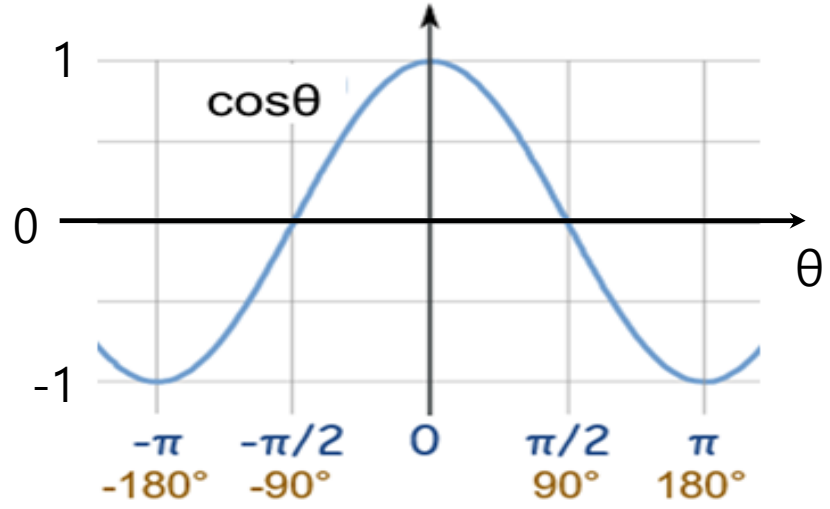
- 두 벡터의 방향이 유사한 정도는 두 벡터 사이의 각 (사이각)을 이용해서 표현
- 사이각이 작을수록 (0에 가까울수록) 유사한 방향성
- 사이각이 클수록 (180도에 가까울수록) 반대 방향
- **코사인 함수**를 통해 수치적 표현

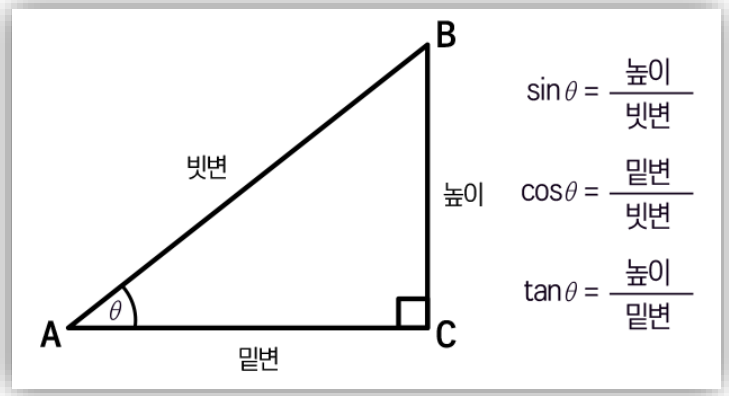


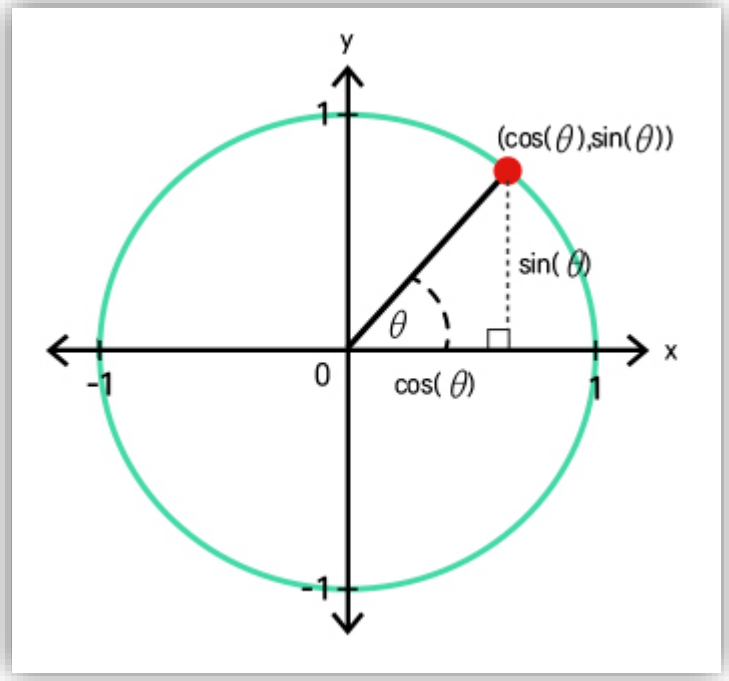
Vector, Vectorization

벡터 (vector) – 유사도 **1 - 방향성 기준**

cosine 함수 직각 삼각형의 빗변에 대한 인접한 변의 비율







Vector, Vectorization

벡터 (vector) – 유사도 **1 - 방향성 기준**

cosine 유사도 – dot product 을 이용한 유사도 구하기

- **내적 연산 (dot product)**
  - $\vec{x} = (x_1, x_2)$ 와 $\vec{y} = (y_1, y_2)$ 의 내적:

    $\vec{x} \bullet \vec{y} = x_1 y_1 + x_2 y_2$ (같은 자리에 있는 원소들을 곱해서 더함)

    직교 좌표계 방법

  - ex, $\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \vec{b} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

    $\vec{a} \bullet \vec{b} = (1 \times 2) + (2 \times 2) = 2 + 4 = 6$

    np.dot(a, b)

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

Vector, Vectorization

벡터 (vector) – 유사도

1 - 방향성 기준

cosine 유사도 – dot product 을 이용한 유사도 구하기

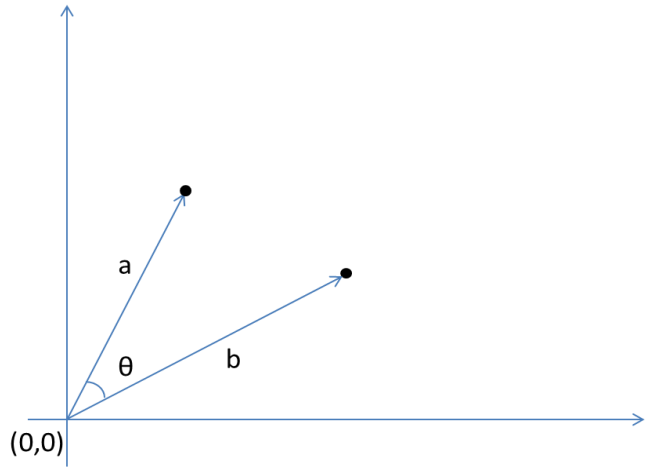
- **내적 연산 (dot product)**
  - $\mathbf{a} \cdot \mathbf{b} = |a||b|\cos\theta$

    기하학적 방법

    |a| : a 벡터의 길이,

    θ : a 벡터와 b 벡터 사이의 각

    $\cos\theta = a \cdot b/|a||b|$ **<-- cosine similarity**



코사인 거리(cosine distance) = 1 – cosθ

Vector, Vectorization

벡터 (vector) – 유사도

1 - 방향성 기준

cosine 유사도 – dot product 을 이용한 유사도 구하기

연습문제

$$\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \vec{b} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Q. a,b 벡터의 코사인 유사도는?

Vector, Vectorization

벡터 (vector) – 유사도

1 - 방향성 기준

cosine 유사도 – dot product 을 이용한 유사도 구하기

연습문제 $\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}, \vec{b} = \begin{bmatrix}2\\2\end{bmatrix}$

Q. a,b 벡터의 코사인 유사도는?

```
a = [1,2]
b = [2,2]
np.dot(a, b)
```

```
aa = np.dot(a, b)
bb = np.linalg.norm(a)
cc = np.linalg.norm(b)
```

```
aa/(bb*cc)
```

```
np.linalg.norm(a)
np.linalg.norm(b)
```

```
yyy = np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b))
1-yyy
```

```
import scipy.spatial.distance as dst
dst.cosine(a,b)
```

Vector, Vectorization

벡터 (vector) – 유사도

1 - 방향성 기준

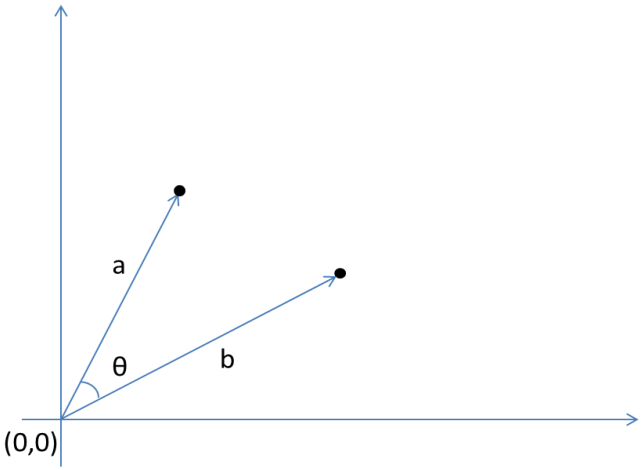
cosine 유사도 – dot product 을 이용한 유사도 구하기

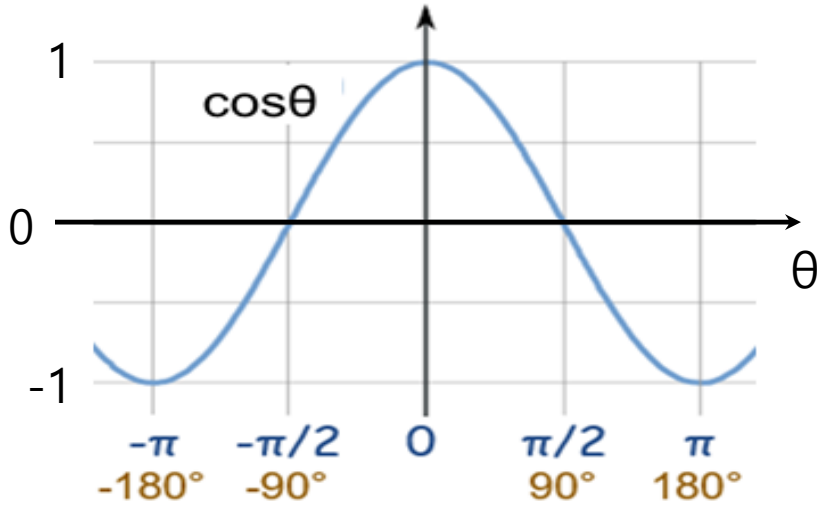
연습문제 $\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}, \vec{b} = \begin{bmatrix}2\\2\end{bmatrix}$

Q. a,b 벡터의 코사인 유사도는?

결과 해석

cosine similarity : 0.9487
cosine distance : 0.0513





-1 : 정확히 반대
0 : 직교
1 : 정확히 같은 방향

Vector, Vectorization

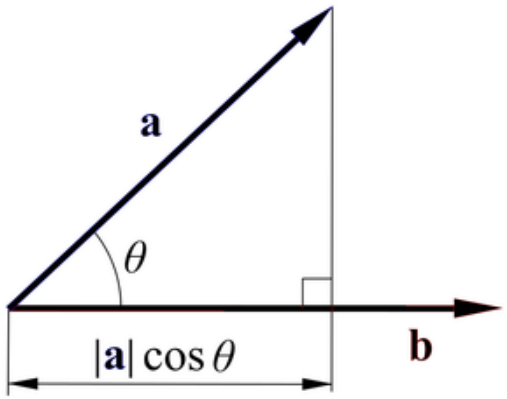
벡터 (vector) – 유사도

cosine 유사도 구하기 – dot product

내적 | 內積 | inner product

적은 '쌓는다'는 뜻의 한자이고, 여기서는 '곱한다'는 뜻이다. 벡터의 곱하기는 두 가지 정의가 있는데, 내적은 벡터를 마치 수처럼 곱하는 개념이다.

벡터에는 방향이 있으므로, 방향이 일치하는 만큼만 곱한다. 예를 들어 두 벡터의 방향이 같으면, 두 벡터의 크기를 그냥 곱한다. 두 벡터가 이루는 각이 90도일 땐, 일치하는 정도가 전혀 없기 때문에 내적의 값은 0이다. 내적은 한 벡터를 다른 벡터로 정사영 시켜서, 그 벡터의 크기를 곱한다.



내적의 기호는 가운데 점을 찍는 것(·)이고, 벡터의 크기를 절대값으로 표시하면, 내적의 값은 다음과 같다.

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| cos\,\theta$$

Vector, Vectorization

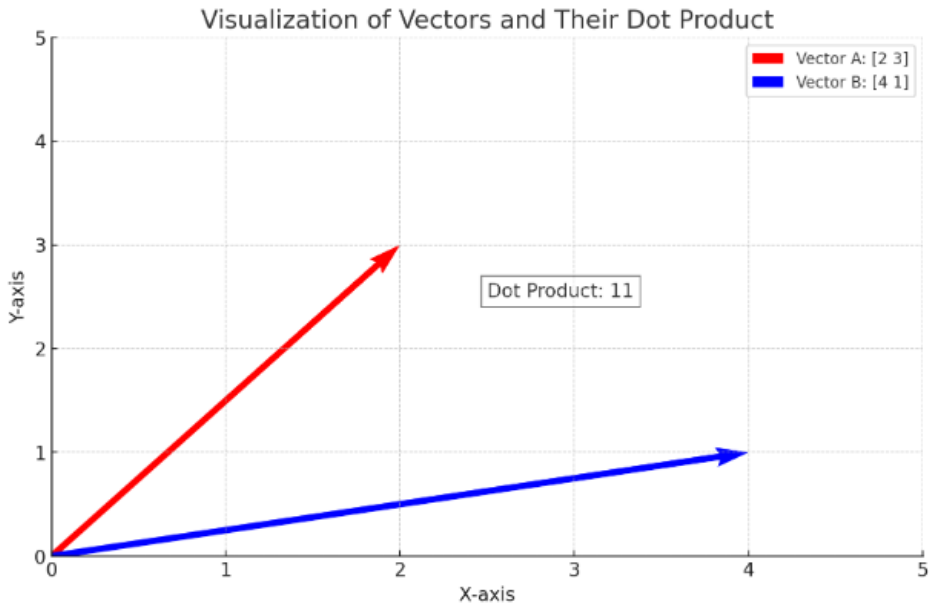
벡터 (vector) – 유사도

cosine 유사도 구하기 – dot product

실습

5.0.vector.distancs.ipynb

5.0.Vector_np_example.ipynb



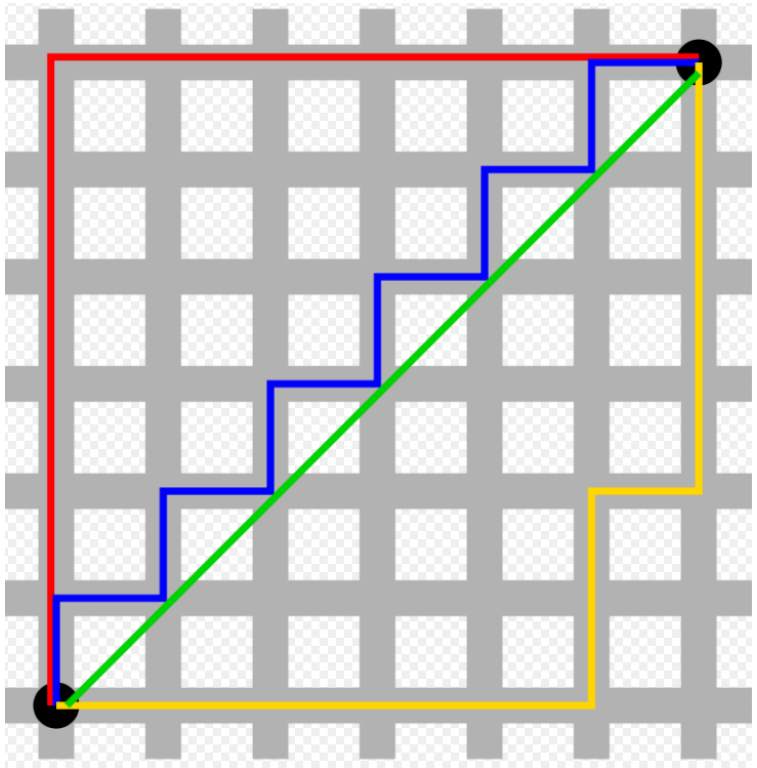
Vector, Vectorization

벡터 (vector) – 유사도

2 - 거리 기준

유클리디안 거리, 맨하튼 거리

: 벡터들이 공간상에서 얼마나 가까이 또는 멀리 떨어져 있는지에 대한 수치적 표현



유클리디안 거리 : $d(\vec{a},\vec{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$

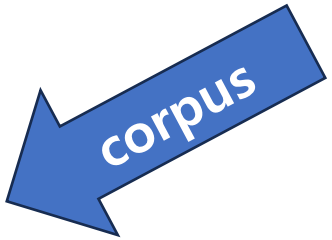
맨하튼 거리 : $d(\vec{a},\vec{b}) = \sum_{i=1}^{n}|a_i - b_i|$

Vectorization

Vector, Vectorization

벡터화 – 문서의 벡터 표현

- ex,
  - Doc 1: 'banana apple apple orange'
  - Doc 2: 'apple carrot eggplant carrot'
  - Doc 3: 'banana mango orange orange'



문서에 사용된 모든 단어 : 'apple', 'banana', 'carrot', 'eggplant', 'mango', 'orange'

- **Vectorization**
  - 각 문서를 **문서에 사용된 단어들로 구성이 된 vector로 표현**
  - 문서들의 vector크기(차원)는 동일 (=전체 단어의 수, **corpus**)
  - Vector의 각 element(원소) 값
    -> Frequency (사용빈도): 각 단어가 각 문서에서 몇 번 사용되었는가?
    -> TF-IDF

Vector, Vectorization

벡터화 – 문서의 벡터 표현

- Doc 1: 'banana apple apple orange'
- Doc 2: 'apple carrot eggplant carrot'
- Doc 3: 'banana mango orange orange'
- all words: 'apple', 'banana', 'carrot', 'eggplant', 'mango', 'orange'

- **frequency**
  - 각 문서를 단어들의 **출현빈도 정보** 사용
  - 순서
    - 전체 데이터에서 사용된 단어들을 알파벳 순으로 배열
    - 각 단어들이 각 문서에서 사용된 횟수 측정

	apple	banana	carrot	eggplant	mango	orange
Doc 1	2	1	0	0	0	1
Doc 2	1	0	2	1	0	0
Doc 3	0	1	0	0	1	2

DTM, document-term matrix

  - but, 각 단어가 해당 문서에서 갖는 **상대적 중요성**을 표현하지 못함

Vector, Vectorization

벡터화 – 문서의 벡터 표현

- **frequency**
  - DTM의 각 행 = 각 문서의 **벡터**

✓ Doc1 = (2,1,0,0,0,1)

✓ Doc2 = (1,0,2,1,0,0)

✓ Doc3 = (0,1,0,0,1,2)

	apple	banana	carrot	eggplant	mango	orange
Doc 1	2	1	0	0	0	1
Doc 2	1	0	2	1	0	0
Doc 3	0	1	0	0	1	2

- **문서들 간의 유사도?**

Vector, Vectorization

벡터화 – TF-IDF

- **TF-IDF (inverse document frequency)**

  단어가 문서 內 에서 지니는 **상대적 중요성** 표현 **상대적 중요성???**

  - ex, 아래 두개의 문서에 표현된 두개의 단어 중 어떠한 단어가 문서 2의 특성을 더 잘 반영?
    이를 어떻게 수치로 표현?

	word1	word2
Doc 1	10	0
Doc 2	10	10

Vector, Vectorization

벡터화 – TF-IDF

- **TF-IDF (inverse document frequency)**
  - 특정 단어가 특정 문서의 uniqueness를 얼마나 나타내는가를 계산하기 위해 사용
  - TF-IDF가 높을수록 해당 단어는 다른 문서에서는 적게 사용, 해당 문서에서 많이 사용 됨
  - 해당 단어가 해당 문서의 uniqueness를 더 많이 표현

- **IDF (inverse document frequency)**
  - https://en.wikipedia.org/wiki/Tf%E2%80%93idf#Inverse_document_frequency
  - 다른 문서에서 얼마나 **사용되지 않았는지**를 의미
  - 1/DF

Vector, Vectorization

벡터화 – TF-IDF

- **IDF (inverse document frequency)**
  - 1/DF
  - IDF
    - 전체 문서들 가운데 **해당 문서를 제외한 나머지 문서들 중**에서 해당 단어가 몇 개의 문서에 사용되었는지를 의미
    - ex) 문서 A에서의 [단어1]에 대한 IDF
      - 데이터셋에 존재하는 전체 문서의 수 = 10
      - if, 문서 A를 제외한 4개의 문서에서 [단어1]사용
        - then, 문서 A에서의 [단어1] DF = 4
        - **IDF = ¼**

Vector, Vectorization

벡터화 – TF-IDF

- Q
  - Frequency?
  - TF?
  - DF?
  - IDF?
  - TF-IDF

Vector, Vectorization

벡터화 – TF-IDF

review

TF

	word1	word2
Doc 1	10	0
Doc 2	**10**	**10**

DF -해당 단어가 사용된 다른 문서의 수

	word1	word2
Doc 1	1	1
Doc 2	1	0

IDF = 1/(DF+1)

	word1	word2
Doc 1	1/2	1/2
Doc 2	1/2	1

TF-IDF

	word1	word2
Doc 1	10*1/2=5	0*1/2=0
Doc 2	10*1/2=**5**	10*1=**10**

Vector, Vectorization

벡터화 – TF-IDF

실습

5.0.doc_vectorization_example1.ipynb (토이 데이터)

5.0.doc_vectorization_example2.ipynb (실 데이터)

Vectorization?

Vector, Vectorization

- 범주형 변수 수치화

- encoding 값(ASCII)

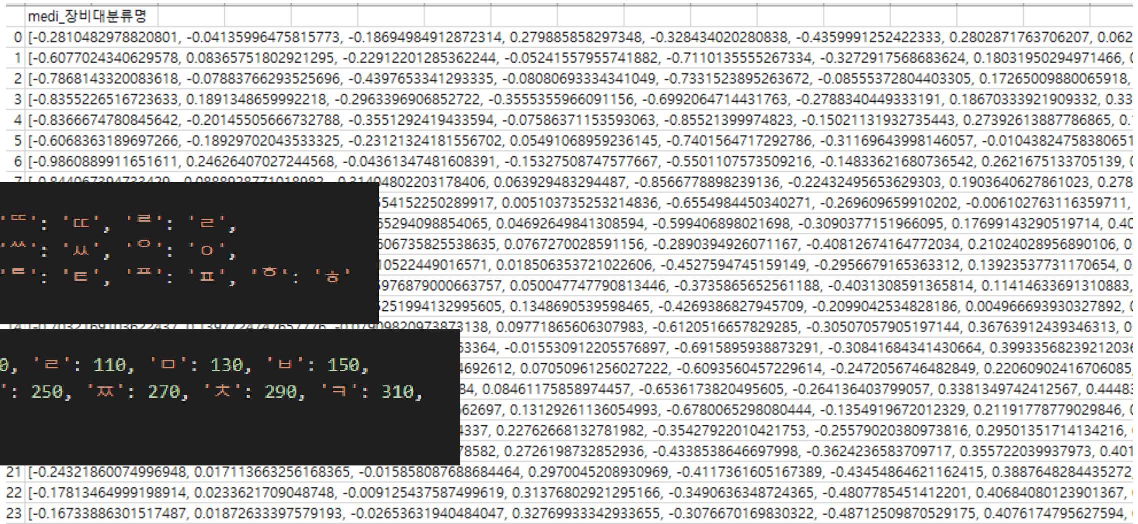
```
def ord(__c: str | bytes | bytearray) -> int: ...
```

속성명	단어	첫번째 글자 ASCII 코드값	두번째 글자 ASCII 코드값	합
medi_시군구명1	치과방사선파노라마장치	52824	44284	57252.4
medi_시군구명1	서울돈화문국악당	49436	50872	54523.2
medi_시군구명1	초음파영상진단기	52488	51020	57590
medi_시군구명1	안성시	50504	49457	55449.7
medi_시군구명1	봄날아트홀	48388	45216	52909.6
medi_시군구명1	1관	49	44288	4477.8
medi_시군구명1	고려대학교의과대학부속구로	44256	47140	48970
medi_시군구명1	금정구푸드뱅크	44552	51221	49674.1
medi_시군구명1	더페인터즈전용관	45908	54168	51324.8
medi_시군구명1	담양군기초푸드뱅크	45812	50577	50869.7
medi_시군구명1	평창군	54217	52285	59445.5
medi_시군구명1	의료법인	51032	47308	55762.8
medi_시군구명1	행도의료재단	54665	46020	59267
medi_시군구명1	해동병원	54644	46041	59248.1
medi_시군구명1	서산카리타스농수산물지원센	49436	49328	54368.8

Vector, Vectorization

- **범주형 변수 수치화**

- Bert 임베딩




```
initial_map = {
    'ㄱ': 'ㄱ', 'ㄲ': 'ㄲ', 'ㄴ': 'ㄴ', 'ㄷ': 'ㄷ', 'ㄸ': 'ㄸ', 'ㄹ': 'ㄹ',
    'ㅁ': 'ㅁ', 'ㅂ': 'ㅂ', 'ㅃ': 'ㅃ', 'ㅅ': 'ㅅ', 'ㅆ': 'ㅆ', 'ㅇ': 'ㅇ',
    'ㅈ': 'ㅈ', 'ㅉ': 'ㅉ', 'ㅊ': 'ㅊ', 'ㅋ': 'ㅋ', 'ㅌ': 'ㅌ', 'ㅍ': 'ㅍ', 'ㅎ': 'ㅎ'
}
```

```
ganada_dict = {
    'ㄱ': 10, 'ㄲ': 30, 'ㄴ': 50, 'ㄷ': 70, 'ㄸ': 90, 'ㄹ': 110, 'ㅁ': 130, 'ㅂ': 150,
    'ㅃ': 170, 'ㅅ': 190, 'ㅆ': 210, 'ㅇ': 230, 'ㅈ': 250, 'ㅉ': 270, 'ㅊ': 290, 'ㅋ': 310,
    'ㅌ': 330, 'ㅍ': 350, 'ㅎ': 370
}
```

- alphabetic order

단어	순서값	단어	순서값	단어	순서값
공간222	11	경성	20	뉴성민병원	60
공간	11	곡성군	20	내시경	60
국군수도병원	11	강서푸드뱅크·마켓	20	나사렛국제병원	60
기기(전극	11	경상국립대학교병원	20	나사렛종합병원	60
기기	11	광산구	20	녹색병원	60
고강도집속형초음파수술기	11	고성군푸드뱅크	20	녹산의료재단동수원병원	60
경기광역기부식품등지원센터북부물류	11	검사기	20	노원어울림극장	62
경기도의료원의정부병원	11	검사기(QCT)	20	나온씨어터	62
공간더하기	11	강서구동지푸드마켓	20	나은병원	62
경기광역기부식품등지원센터남부물류	11	강서푸드뱅크마켓	20	노을소극장	62
기관지경	11	검사기(Dual-Energy	20	남원시사회복지협의회기초푸드뱅크	62
경기도광역푸드뱅크	11	괴산군푸드뱅크	20	남양주시	62
경기도의료원	11	금산군푸드뱅크	20	노원구	62
건국대학교병원	11	군산시기초푸드뱅크	20	남양주나눔병원	62
기기(마그네틱	11	검사기(QUS)	20	남원시	62
(언더스테이지)	12	권선구	20	노원문화예술회관	62
(의)나사렛의료재단	12	광선요법기	20	남양주	62

Vectorization in Deep learning

Vector, Vectorization

Vectorization(벡터화)

- **토큰화**

  : 문장을 **단어**별로 나누는 것

  (한글) 형태소를 분석하고 나누는 것

  :개별 단어로 분리하여 텍스트 데이터의 기본적인 **빌딩 블록**으로 사용

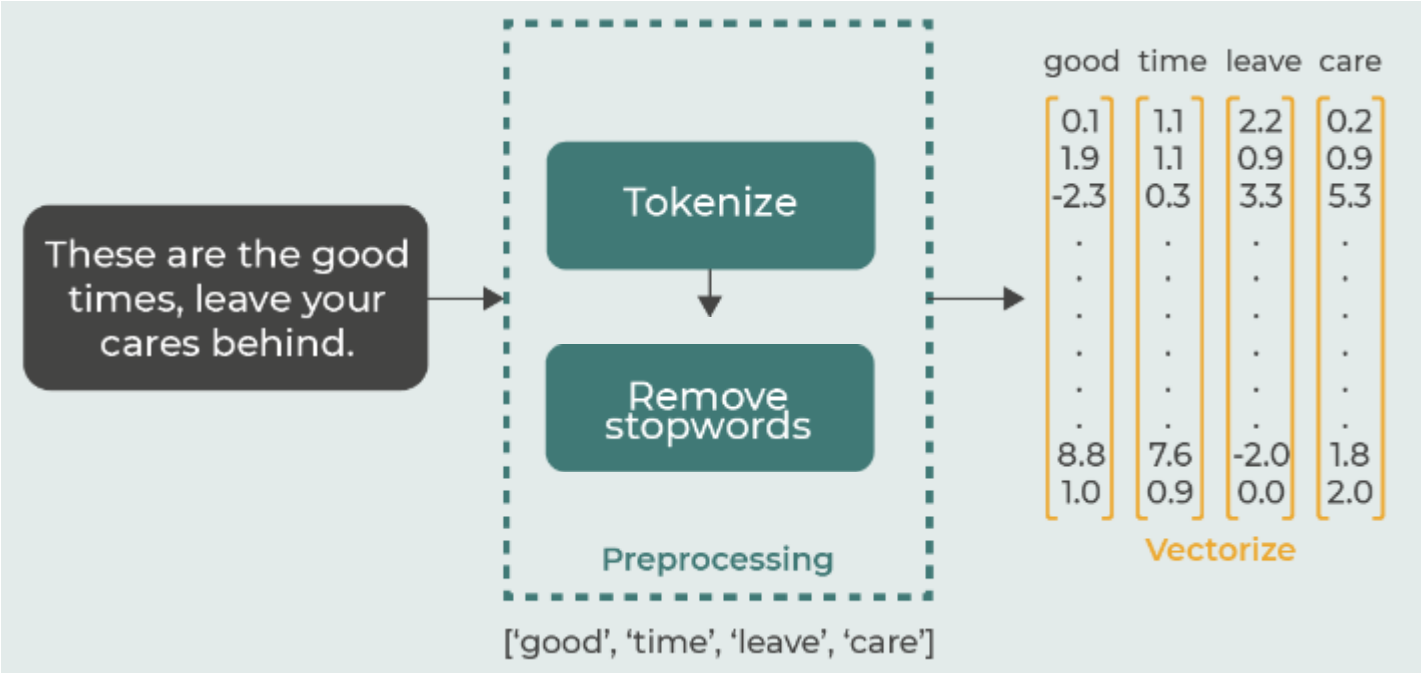
Vector, Vectorization

Vectorization(벡터화)

- **벡터화**

  : 단어를 추출하여 **수치화**

  : 토큰화 된 텍스트를 숫자의 형태, 즉 **벡터로 변환**하는 과정



https://openclassrooms.com/en/courses/6532301-introduction-to-natural-language-processing/8081284-apply-a-simple-bag-of-words-approach

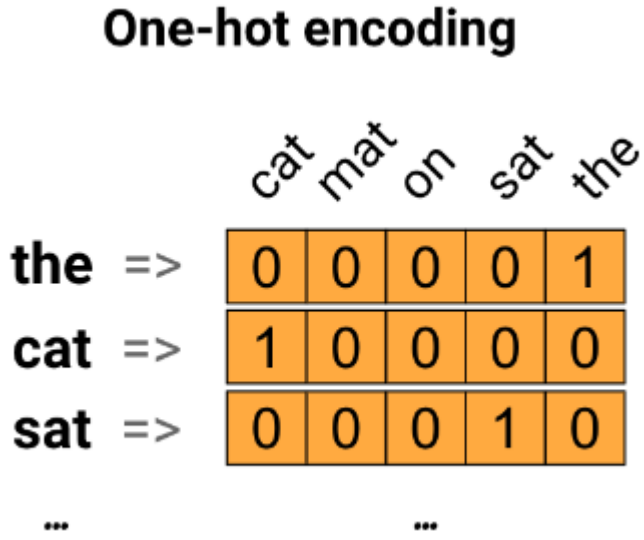
Vector, Vectorization

Vectorization(벡터화)

- **벡터화 - One-hot encoding**

  : 각 단어를 하나의 인덱스가 1이고 나머지는 0인 벡터로 표현

	bed	cat	dog	face	my	on	sat	the
문서 A	0	1	0	1	1	1	1	1
문서 B	1	0	1	0	1	1	1	1



cf) 단어 임베딩(Word Embedding)

> 각 단어를 고정된 크기의 실수 벡터로 표현

> 단어 간의 의미적 관계 포착

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

5.01.IMDB.embeding.ipynb

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

1. IMDB 데이터셋 로딩

```
from tensorflow.keras.datasets import imdb
max_features = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
```

- Keras를 사용하여 IMDB 영화 리뷰 데이터셋 로드
- max_features = 빈도수 기준 상위 10,000개의 단어만 사용

  -> 리뷰 텍스트 벡터화 시 단어 인덱스의 범위가 1부터 10,000까지로 제한

- 각각 리뷰 텍스트가 정수 시퀀스(단어 인덱스)로 변환 되어 로드

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

1. IMDB 데이터셋 로딩

- 각각 리뷰 텍스트가 **정수 시퀀스(단어 인덱스)로 변환** 되어 로드

```
 1 len(x_train[0])
```

```
218
```

```
 1 print(x_train[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66,
```

```
<START> this film was just brilliant casting location scenery story direction
```

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

2. 데이터 전처리: 패딩

```
from tensorflow.keras.preprocessing import sequence
max_len = 200 # 문서 길이를 200 단어로 설정
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
```

- 패딩(padding): 리뷰 데이터 길이 통일
- max_len = 200: 리뷰의 최대 길이를 200으로 설정
- 리뷰 길이 < 200: 부족 부분을 0으로 패딩
  리뷰 길이 > 200: truncate

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

2. 데이터 전처리: 패딩

- 리뷰 길이 < 200: 부족 부분을 0으로 패딩
  리뷰 길이 > 200: truncate

```
1 x_train[1]
```

```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          1,  194, 1153,  194, 8255,   78,  228,    5,    6, 1463, 4369,
       5012,  134,   26,    4,  715,    8,  118, 1634,   14,  394,   20,
```

```
1 len(x_train[0])
```

```
218
```

```
1 len(x_train[0])
```

```
200
```

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

3. vectorization

```
model_fnn.add(Embedding(max_features, 64, input_length=max_len)) # 임베딩 층 추가
```

vectorization

- Fnn 모델에 임베딩 층(Embedding layer) 추가
- **숫자로 인코딩 된 단어를 받아 각 단어를 64차원의 벡터로 변환**
- input_length=max_len: 모델이 입력으로 받을 특성의 수(리뷰 당 단어의 수 200개)

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

3. vectorization

```
model_fnn.add(Embedding(max_features, 64, input_length=max_len)) # 임베딩 층 추가
```

- **임베딩 층(Embedding layer)**

1. **입력**: **숫자** 인덱스(ex 단어 'cat' -> 2로 매핑)로 표현된 단어들의 시퀀스를 입력으로 받음

   숫자: 사전에 정의된 어휘(vocabulary)의 인덱스

2. **벡터화**: 숫자 인덱스를 (밀집) 벡터로 변환

   초기값: 랜덤 초기화, 모델 **학습을 통해 업데이트**(최적화) 됩니다. -> 단어 간의 의미 관계 반영

   (올바른 예측을 위해) 단어 벡터 간의 거리 조정 -> 의미적으로 유사한 단어들이 벡터 공간에서 가까워지도록 update

3. **출력**: 벡터 출력

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 200, 64)           640000
```

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

3. vectorization

```
1 reverse_word_index[4]
```

```
'the'
```

```
9 embedding_layer_weights[4]
```

```
Embedding layer weights shape: (10000, 64)
Initial vector for the first word:
 [-0.00259084  0.01449068 -0.0360497   0.04829049  0.03736104  0.0439479
  0.04184875  0.03720633 -0.00947944 -0.02207879  0.01567849  0.01323043
 -0.00623485 -0.02969201  0.02015174  0.0419557   0.03964961  0.03541067
 -0.04929231 -0.02532177 -0.01945633  0.0294967  -0.02734786 -0.00235792
  0.04007456  0.01345647 -0.02578268  0.02072446 -0.01814718 -0.00438038
 -0.00013449  0.04992953  0.00172102 -0.02740901 -0.04531569  0.01248584
  0.03048811 -0.01235138 -0.03553003 -0.0029698  -0.00998558 -0.0273586
 -0.01493003 -0.00096365 -0.0060817   0.01747804 -0.00963595  0.04506803
 -0.02521282 -0.04632554 -0.0016393  -0.02918473  0.02854191 -0.00418777
 -0.01391535 -0.01575425 -0.00093541 -0.00559113 -0.02639507  0.0006377
  0.01263886 -0.00140343  0.00334054  0.03750087]
```

Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

3. vectorization

```
9 embedding_layer_weights[4]

Embedding layer weights shape: (10000, 64)
Initial vector for the first word:
 [-0.00259084  0.01449068 -0.0360497   0.04829049  0.03736104  0.0439479
  0.04184875  0.03720633 -0.00947944 -0.02207879  0.01567849  0.01323043
 -0.00623485 -0.02969201  0.02015174  0.0419557   0.03964961  0.03541067
 -0.04929231 -0.02532177 -0.0194
  0.04007456  0.01345647 -0.0257
 -0.00013449  0.04992953  0.0017
  0.03048811 -0.01235138 -0.0355
 -0.01493003 -0.00096365 -0.0060
 -0.0252128   .046
 -0.0139153   .015
  0.0126388   .001
```

모델 학습 후
임베딩 벡터값 최적화

```
Initial vector for the first word:
 [-0.01129117 -0.01072699 -0.02890531  0.06105558  0.01488835  0.04205501
  0.07614601  0.05581363  0.00186577 -0.02362371 -0.013225    0.01234909
 -0.03701784 -0.03587553  0.0205195   0.09616958  0.06541682  0.02706315
 -0.05288105 -0.05047932 -0.03727179  0.05876546 -0.01837556  0.00148083
             0.04694126 -0.02880276  0.05579894 -0.04578283  0.0403077
             0.0854268   0.00199568 -0.02123355 -0.04416034 -0.01106741
            -0.02136315 -0.03137319 -0.00932574 -0.01544816 -0.01655687
 -0.02143353 -0.03304139 -0.01574039  0.0116671  -0.00489496  0.07434656
 -0.02719317 -0.07976457 -0.01711885 -0.01896027  0.0396108  -0.01163105
 -0.03058972 -0.01853502 -0.04094128  0.01945106 -0.0372944   0.01734378
  0.00567869 -0.00224807  0.02043356  0.04622691]
```

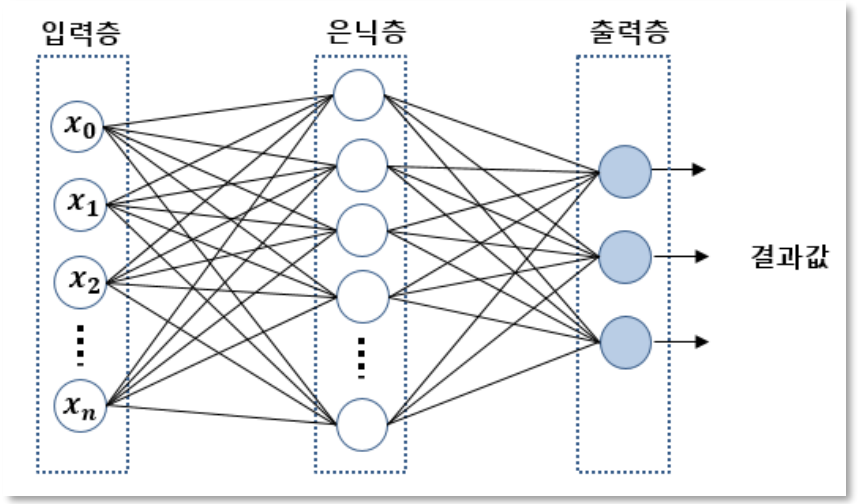
Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

4. Flatten

```
model_fnn.add(Flatten())
```

- Dense 층으로 데이터 전달을 위해 일렬로 펼치는(Flatten) 과정
- 임베딩 층: 각 리뷰가 200개의 단어로 구성. 각 단어는 64차원의 벡터로 표현
- 임베딩 층에서 생성된 벡터를 Fully Connected Layer에 전달하기 위해 2차원의 임베딩 출력을 1차원으로 변환



Vector, Vectorization

Vectorization(벡터화) – IMDB영화평

4. Flatten

```
model_fnn.add(Flatten())
```

- 2차원의 임베딩 출력(200개 단어 * 64 차원 벡터)을 1차원으로 변환

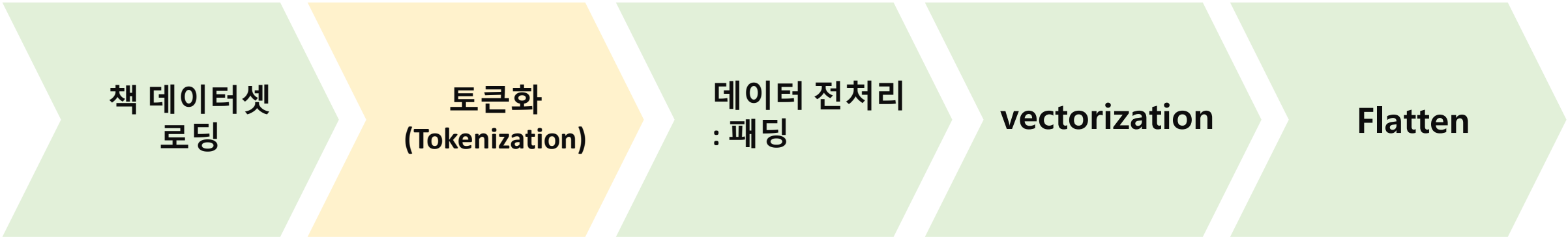
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 12800)             0
```

```
[[ 0.0410122  -0.01066505 -0.00351053 ...  0.01387748 -0.03767189
   0.03997559]]
Shape of the output after Flatten layer: (1, 12800)
```

Vector, Vectorization

Vectorization(벡터화) – y24

5.01.y24.embeding.ipynb



Vector, Vectorization

Vectorization(벡터화) – Y24

1. 책 데이터셋 로딩

```
df_1 = pd.read_csv('/content/book.csv', encoding='euc-kr')
```

```
X = df['Title']
```

y24.modeling.ipynb

Vector, Vectorization

Vectorization(벡터화) – Y24

2. 토큰화 (Tokenization)

```
1 # 텍스트 토큰화 및 시퀀스 변환
2 tokenizer = Tokenizer()
3 # Tokenizer 객체를 사용하여 텍스트를 토큰화, 각 토큰(단어)에 고유한 정수 인덱스 할당
4 tokenizer.fit_on_texts(X)
5 X_seq = tokenizer.texts_to_sequences(X)
```

각 책 제목을 구성하는 단어들을 해당 인덱스로 변환, 시퀀스 형성

- Tokenizer(): 텍스트를 토큰화 하기 위한 객체. **각 단어를 고유한 정수 인덱스에 매핑**
- fit_on_texts(X): 책 제목 데이터(X)를 대상으로 토큰화 수행

  fit_on_texts: Tokenizer 객체가 전달된 텍스트에서 모든 고유 단어를 찾아 각 단어에 고유한 정수 할당
- texts_to_sequences(X): 각 텍스트(책 제목)를 fit_on_texts 메서드에서 생성한 단어 인덱스를 사용,

  숫자(정수) 시퀀스로 변환

Vector, Vectorization

Vectorization(벡터화) – Y24

2. 토큰화 (Tokenization)

각 책 제목을 구성하는 단어들을 해당 인덱스로 변환, 시퀀스 형성

```
1 print(X.iloc[0])
2
```

강철왕국 프로이센

```
1 X_seq[0]
```

[9503, 9504]

Vector, Vectorization

Vectorization(벡터화) – Y24

3. 패딩

```
1 # 패딩으로 시퀀스 길이 맞추기
2 X_pad = pad_sequences(X_seq, maxlen=30)
```

```
1 X_pad[0]
```

```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0, 9503, 9504], dtype=int32)
```

Vector, Vectorization

Vectorization(벡터화) – Y24

4. vectorization

```
model_fnn.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=128, input_length=30))
```

```
Embedding layer weights shape: (25496, 128)
Initial vector for the first word:
 [-0.03981066 -0.01941463  0.0319162   0.00764103  0.01295251 -0.03474094
 -0.00250152 -0.01514218 -0.04240872 -0.00389884 -0.0138077   0.01345051
 -0.03113945 -0.00014126  0.02585062 -0.00438229  0.01037584 -0.0176952
  0.04108013 -0.03314181 -0.0065045   0.04581195  0.03592794  0.03832057
 -0.00471351 -0.03744949 -0.04682071 -0.03534354  0.04075095  0.01589939
 -0.01536893 -0.00907686  0.00739302  0.03755239  0.03539229  0.03527451
 -0.00766027 -0.0070986   0.00286472  0.04461044 -0.03298272 -0.00072401
  0.04269766 -0.02962944  0.01233011 -0.01376265  0.01036043 -0.00104185
  0.04164011 -0.04589572 -0.00240383 -0.03573263  0.03195244 -0.000928
 -0.00478796 -0.04303749 -0.0104311   0.00057267 -0.04817669  0.02588532
  0.01045078  0.04246173 -0.00112297  0.01680389  0.03367605 -0.0049134
 -0.04796031 -0.00618925  0.01165988  0.03728602 -0.02074103 -0.00731765
 -0.02441176 -0.02240543 -0.03259652 -0.00865977 -0.03143211  0.01088673
  0.03652524 -0.03892237  0.03708203 -0.02407414 -0.00475363  0.01436189
 -0.03945776  0.01753466  0.02339962 -0.03670602  0.00201236 -0.03067845
  0.01328691 -0.04208748  0.01391247  0.01276967  0.00319422  0.0062793
 -0.02372679 -0.02974217 -0.04262877 -0.04532481 -0.04950618 -0.04666634
  0.00614103  0.04888031 -0.02978773 -0.01146734 -0.03503977 -0.00433663
 -0.02592756 -0.04747767  0.04777292  0.03284706 -0.04805249  0.00999932
 -0.03007211 -0.0326872  -0.00889897 -0.03365169 -0.02563264  0.03577197
 -0.03576284  0.03379333  0.02216517  0.00037787 -0.02404809  0.03084159
 -0.03947303 -0.04638777]
```

Vector, Vectorization

Vectorization(벡터화) – Y24

4. vectorization

```
model_fnn.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=128, input_length=30))
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_3 (Embedding)      (None, 30, 128)           3263488
```

```
1 len(tokenizer.word_index)
```

```
25495
```

128*(25495+1)

Vector, Vectorization

Vectorization(벡터화) – Y24

5. Flatten

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_3 (Embedding)      (None, 30, 128)           3263488

 flatten_1 (Flatten)          (None, 3840)              0

 dense_2 (Dense)              (None, 64)                245824

 dense_3 (Dense)              (None, 1)                 65
```

30 * 128 = 3,840

Vector, Vectorization

Vectorization(벡터화) – Y24

Sum up

```
1 print(X.iloc[0])
2
```

강철왕국 프로이센

```
1 X_seq[0]
```

[9503, 9504]

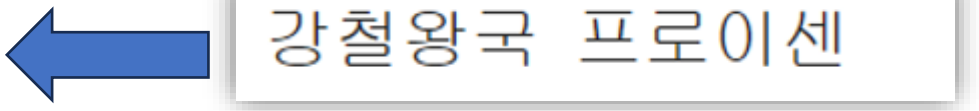
토큰화 (Tokenization) : 각 책 제목을 구성하는 단어들을 해당 인덱스로 변환, 시퀀스 형성

패딩 (Padding) : 시퀀스(각 책 제목)의 길이를 맞춤

```
1 # 패딩으로 시퀀스 길이 맞추기
2 X_pad = pad_sequences(X_seq, maxlen=30)
```

```
1 X_pad[0]
```

```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0, 9503, 9504], dtype=int32)
```



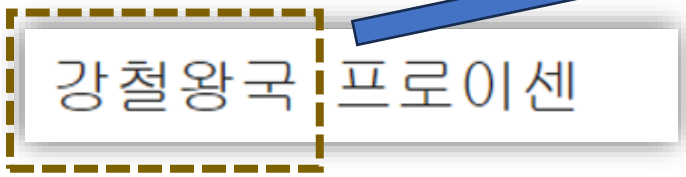
Vector, Vectorization

Vectorization(벡터화) – Y24

Sum up

Vectorization(벡터화) : 각 책 제목을 구성하는 단어들을 숫자로 변환

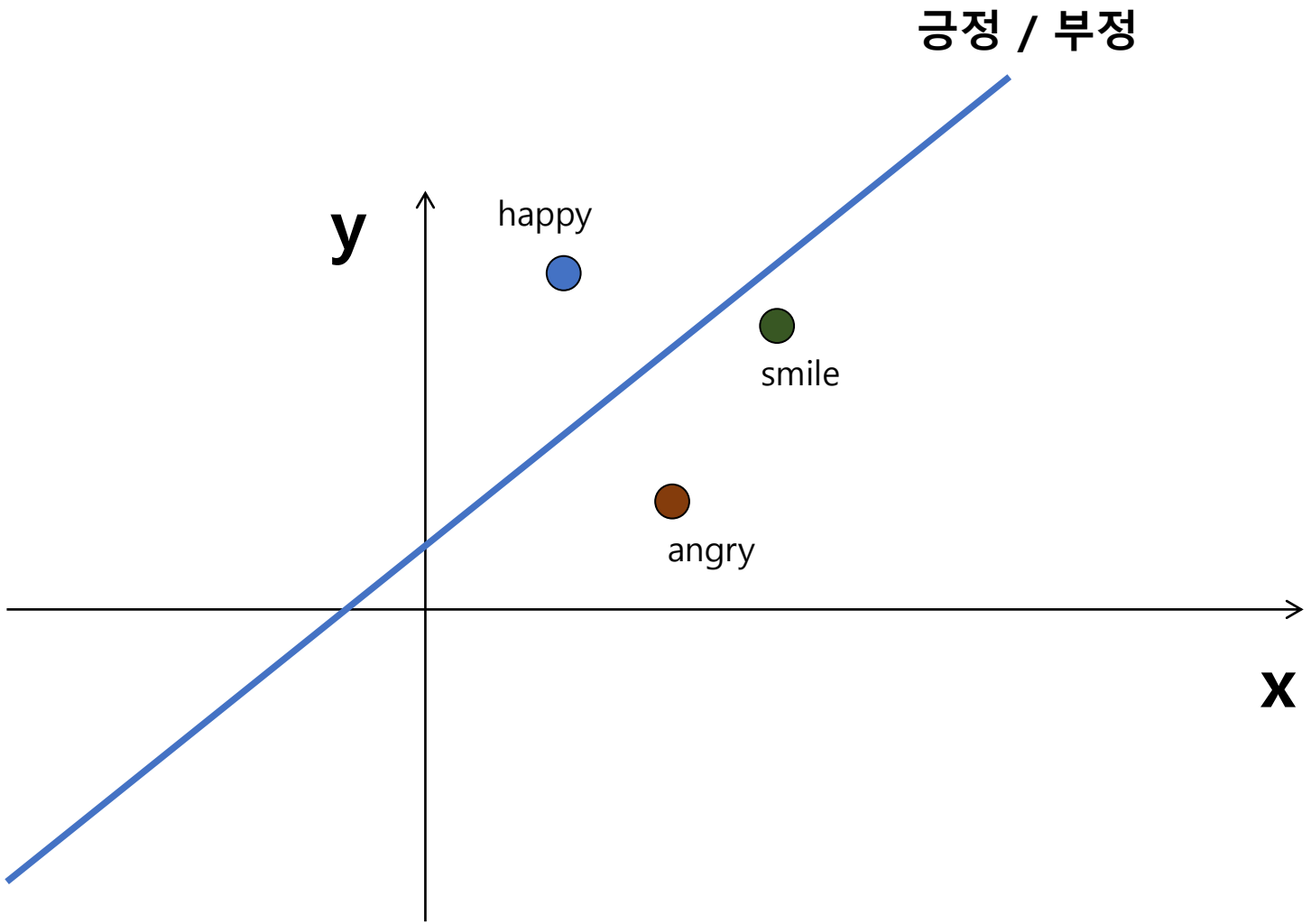
```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0, 9503, 9504], dtype=int32)
```




```
Embedding layer weights shape: (25496, 128)
Initial vector for the first word:
 [-0.03981066 -0.01941463  0.0319162   0.00764103  0.01295251 -0.03474094
 -0.00250152 -0.01514218 -0.04240872 -0.00389884 -0.0138077   0.01345051
 -0.03113945 -0.00014126  0.02585062 -0.00438229  0.01037584 -0.0176952
  0.04108013 -0.03314181 -0.0065045   0.04581195  0.03592794  0.03832057
 -0.00471351 -0.03744949 -0.04682071 -0.03534354  0.04075095  0.01589939
 -0.01536893 -0.00907686  0.00739302  0.03755239  0.03539229  0.03527451
 -0.00766027 -0.0070986   0.00286472  0.04461044 -0.03298272 -0.00072401
  0.04269766 -0.02962944  0.01233011 -0.01376265  0.01036043 -0.00104185
  0.04164011 -0.04589572 -0.00240383 -0.03573263  0.03195244 -0.000928
 -0.00478796 -0.04303749 -0.0104311   0.00057267 -0.04817669  0.02588532
  0.01045078  0.04246173 -0.00112297  0.01680389  0.03367605 -0.0049134
 -0.04796031 -0.00618925  0.01165988  0.03728602 -0.02074103 -0.00731765
 -0.02441176 -0.02240543 -0.03259652 -0.00865977 -0.03143211  0.01088673
  0.03652524 -0.03892237  0.03708203 -0.02407414 -0.00475363  0.01436189
 -0.03945776  0.01753466  0.02339962 -0.03670602  0.00201236 -0.03067845
  0.01328691 -0.04208748  0.01391247  0.01276967  0.00319422  0.0062793
 -0.02372679 -0.02974217 -0.04262877 -0.04532481 -0.04950618 -0.04666634
  0.00614103  0.04888031 -0.02978773 -0.01146734 -0.03503977 -0.00433663
 -0.02592756 -0.04747767  0.04777292  0.03284706 -0.04805249  0.00999932
 -0.03007211 -0.0326872  -0.00889897 -0.03365169 -0.02563264  0.03577197
 -0.03576284  0.03379333  0.02216517  0.00037787 -0.02404809  0.03084159
 -0.03947303 -0.04638777]
```

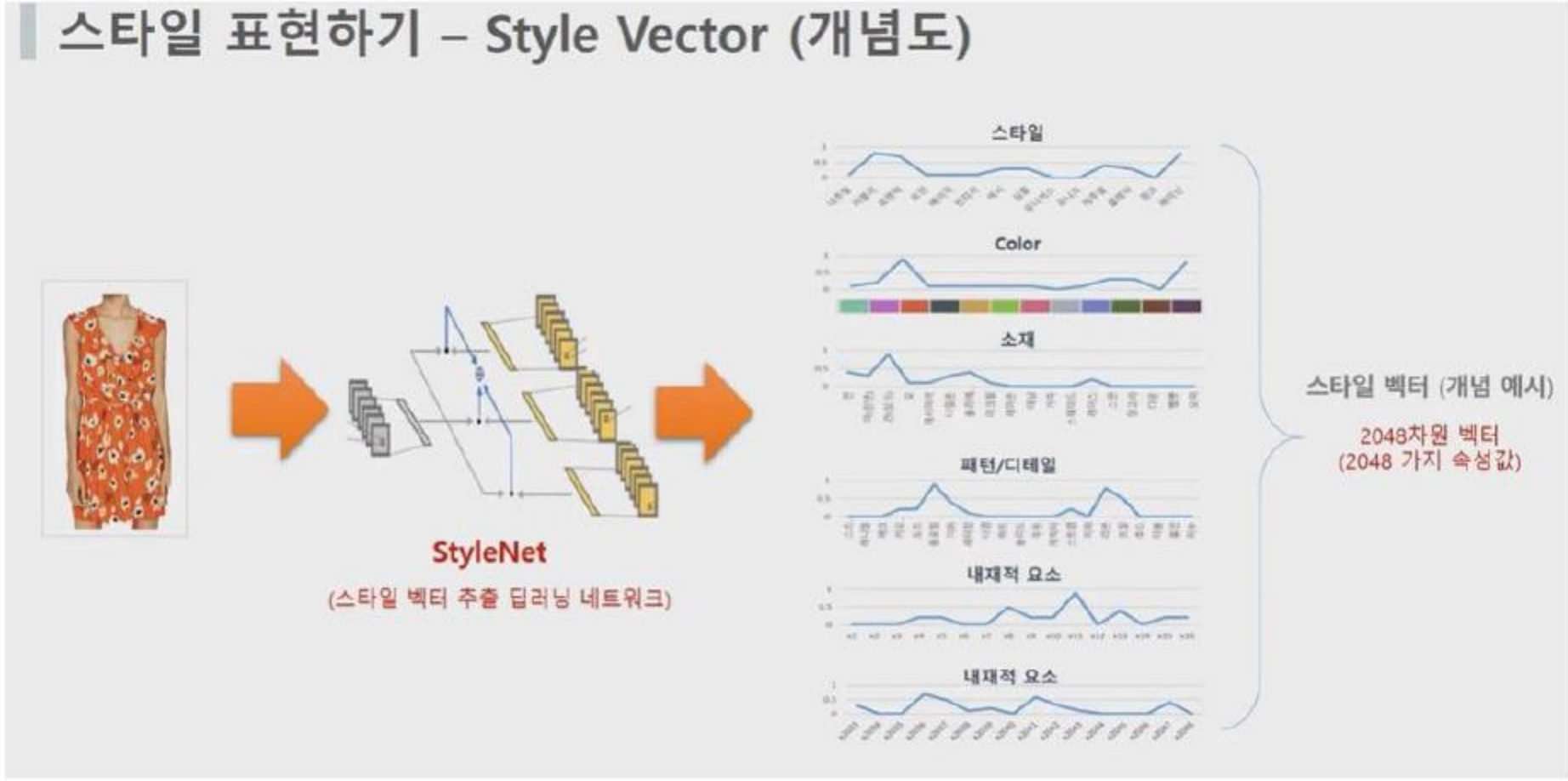
Vector, Vectorization

벡터 (vector) – 학습



What to Vectorization?

Vector, Vectorization

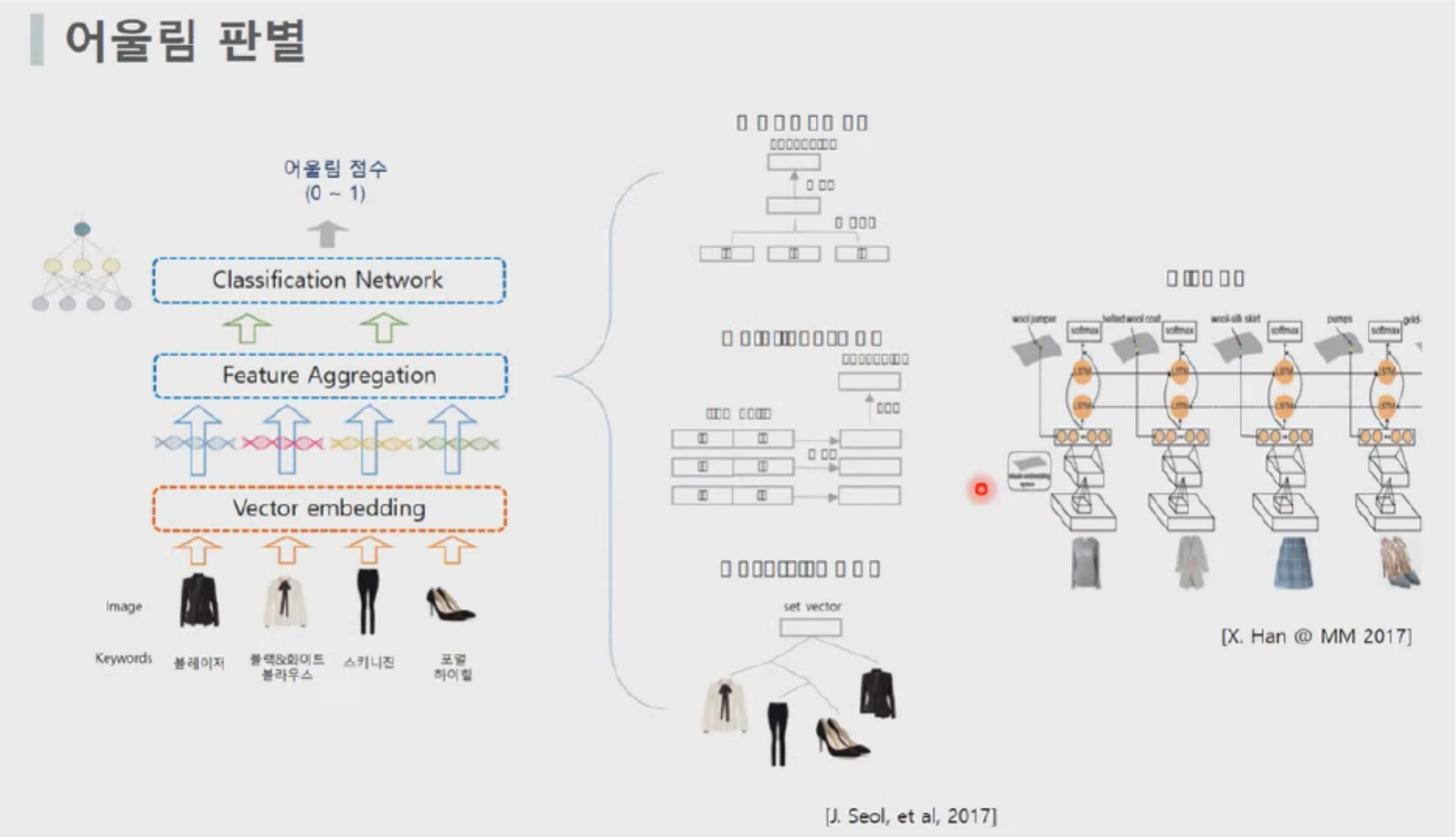


이상구교수, https://www.youtube.com/watch?v=m2U51AXnoC0

Vector, Vectorization

이상구교수, https://www.youtube.com/watch?v=m2U51AXnoC0

Vector, Vectorization



이상구교수, https://www.youtube.com/watch?v=m2U51AXnoC0

THANK YOU