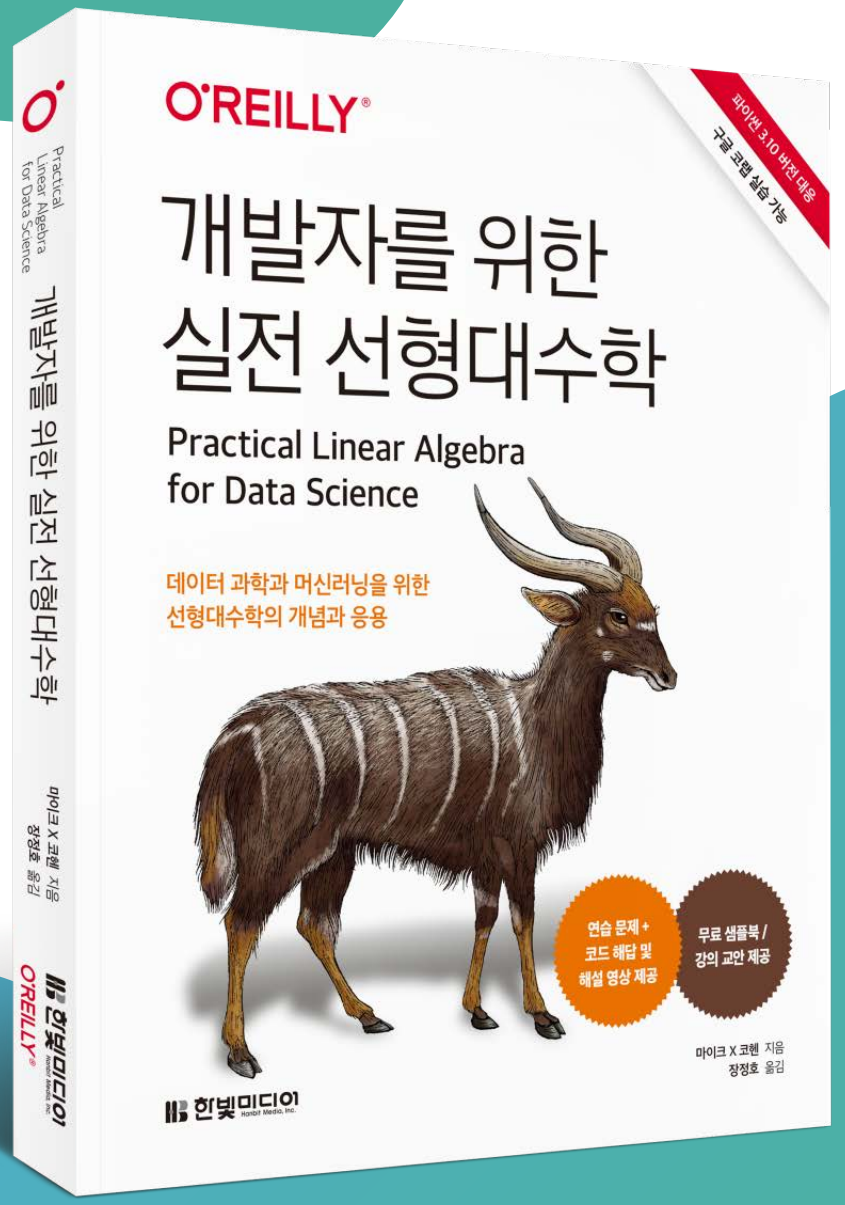
개발자를 위한

실전 선형대수학



시작하기전에

지은이 마이크 X 코헨 Mike X Cohen
네덜란드 라드바우드 대학 메디컬 센터의 돈더스 연구소 소속 신경과학 부교수. 20년 이상 과학 코딩, 데이터 분석, 통계 및 관련 주제를 가르치며, 여러 온라인 강좌와 교과서를 썼습니다.

옮긴이 장정호 egnarzhome@gmail.com
네이버 검색 소프트웨어 엔지니어. 2006년에 티맥스에서 애플리케이션/시스템 간 데이터 전송 시스템 개발을 시작으로, 다음커뮤니케이션에서 데이터 마이닝 업무, SAP에서 칼럼 기반의 인메모리 RDBMS인 HANA 개발에 동참했으며, 그 후 빅데이터 저장/분석 시스템 영역에 관한 연구를 통해 네이버에서 데이터 분석 시스템을 개발을 담당했습니다. 한빛미디어에서 『쿠퍼네티스 모범 사례』(2020), 『하둡 완벽 가이드(4판)』(2017), 『하이브 완벽 가이드』(2013) 등을 번역했습니다.
SK텔레콤의 데이터 플랫폼 엔지니어. 네이버와 SAP에서 데이터 플랫폼과 데이터베이스 개발에 참여했습니다.

- **예제 코드: https://github.com/Sancho-kim/LinAlg4DS**
- **원서의 깃허브 주소: https://github.com/mikexcohen/LinAlg4DataScience**
- **온라인 강의: https://url.kr/xrney7**

이 책의 학습 목표

CHAPTER 01: 벡터, 파트1: 벡터와 벡터의 기본 연산

CHAPTER 02: 벡터, 파트 2: 벡터의 확장 개념

CHAPTER 03: 벡터 응용: 데이터 분석에서의 벡터

CHAPTER 04: 행렬, 파트 1: 행렬과 행렬의 기본 연산

CHAPTER 05: 행렬, 파트 2: 행렬의 확장 개념

CHAPTER 06: 행렬 응용: 데이터 분석에서의 행렬

CHAPTER 07: 역행렬: 행렬 방정식의 만능 키

CHAPTER 08: 직교 행렬과 QR 분해: 선형대수학의 핵심 분해법 1

CHAPTER 09: 행 축소와 LU 분해: 선형대수학의 핵심 분해법 2

CHAPTER 10: 일반 선형 모델 및 최소제곱법: 우주를 이해하기 위한 방법

CHAPTER 11: 최소제곱법 응용: 실제 데이터를 활용한 최소제곱법

CHAPTER 12: 고윳값 분해: 선형대수학의 진주

CHAPTER 13: 특잇값 분해: 고윳값 분해의 다음 단계

CHAPTER 14: 고윳값 분해와 SVD 응용: 선형대수학의 선물

부록 A : 파이썬 튜토리얼

CHAPTER 01: 벡터, 파트1: 벡터와 벡터의 기본 연산

Contents

- CHAPTER 01: 벡터, 파트1: 벡터와 벡터의 기본 연산

SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(1)

- 벡터(vector): 수를 순서대로 나열한 것
- 벡터의 특징
  - 차원(Dimensionality): 벡터가 가진 원소의 수
    - $\mathbb{R}^N$으로 나타내는데, 여기서 $\mathbb{R}$은 실수(Real number)를 N은 차원
    - 예를 들어 두 개의 원소가 있는 벡터는 $\mathbb{R}^2$에 속함
    - $\mathbb{R}$문자는 LaTex 코드를 사용해서 만들지만 R_2이나 R2, R^2로 쓸 수도 있음
  - 방향(orientation): 벡터가 열 방향(높이 세워진)인지 행 방향(길고 평평하게 누운)인지를 나타냄

$$x = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 6 \end{bmatrix}, \quad y = \begin{bmatrix} 3 \\ -7 \end{bmatrix}, \quad z = \begin{bmatrix} 1 & 4 & 5 & 6 \end{bmatrix}$$

- **x는 4차원 열벡터, y는 2차원 열벡터, z는 4차원 행벡터**
- **또는 $x \in \mathbb{R}^4$ 와 같이 쓸 수도 있음**

식 1-1 열벡터와 행벡터의 예

SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(2)

- 파이썬에서 벡터 또는 행렬의 차원
  - 수 객체를 출력하는 데 사용되는 기하학적 차원의 수
  - 모든 벡터는 벡터가 가진 원소의 수(수학적 차원)에 상관없이 NumPy에서 '2차원 배열'로 간주
  - 특정 방향이 없는 수 나열은 원소 수에 상관없이 파이썬에서 1차원 배열
    - 이 배열은 행으로 출력되지만 나중에 나오는 행벡터와 다름
  - 수학적 차원, 즉 벡터의 원소 수는 파이썬에서 벡터의 길이(length) 또는 모양(shape)이라고 함
- 벡터의 표기
  - 일반적으로 벡터 **v**는 진한 로마자 소문자 **v**로 나타냄
    - 또는 이탤릭체(v)를 사용하거나 위쪽에 화살표를 붙임($\vec{v}$)
- 선형대수학에서 보통 벡터에 아무런 표시가 없다면 열 방향이라고 가정
  - 행벡터는 W^T로 표시
  - T는 전치 연산(transpose operation)을 나타냄

SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(3)

- 파이썬에서 벡터를 표현하는 데이터 타입
  - 리스트 타입: 벡터를 표현하는 가장 간단한 방법이지만 선형대수학 응용 분야에서는 잘 사용하지 않음
    - 많은 선형대수학 연산은 파이썬 리스트에 대해 잘 동작하지 않음
  - NumPy 배열
- 벡터를 생성하는 네 가지 방법

```
asList = [1,2,3]
asArray = np.array([1,2,3]) # 1차원 배열
rowVec = np.array([ [1,2,3] ]) # 행
colVec = np.array([ [1],[2],[3] ]) # 열
```

- asArray 변수는 방향이 없는 배열로, 행이나 열벡터가 아니라 NumPy의 숫자 1차원 리스트
- NumPy의 방향은 대괄호로 지정
  - 가장 바깥쪽 대괄호는 모든 숫자를 하나의 객체로 묶음
  - 추가적인 내부 괄호 집합은 행을 나타냄
- 행벡터(변수 rowVec)는 하나의 행이 모든 숫자를 가지지만 열벡터(변수 colVec)는 하나의 숫자를 가진 행이 여러 개

```
print(f'asList: {np.shape(asList)}')
print(f'asArray: {asArray.shape}')
print(f'rowVec: {rowVec.shape}')
print(f'colVec: {colVec.shape}')
```

⇨

```
asList:  (3,)
asArray: (3,)
rowVec:  (1, 3)
colVec:  (3, 1)
```

- 1차원 배열인 asArray는 모양이 (3)이지만 방향이 부여된 벡터는 2차원 배열이며 방향에 따라 모양이 (1,3) 또는 (3,1)
- 차수는 항상 (행, 열)로 표현

SECTION 01-1 NumPy로 벡터 생성 및 시각화하기(4)

1.1.1 벡터 기하학

- 순서대로 나열된 수 목록(ordered list of numbers)
  - 벡터의 대수학적 해석 – 순서대로 나열된 수 목록
  - 기하학적 해석 – 특정 길이(또는 크기(magnitude))와 방향(또는 각도(angle): 양의 x축을 기준으로 계산됨)을 가진 직선
- 기준 위치(standard position)

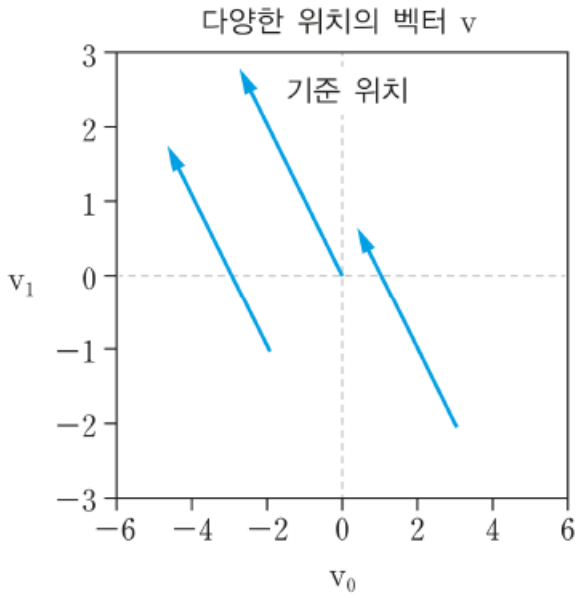


그림 1-1 모든 화살표는 동일한 벡터
기준 위치의 벡터는 꼬리가 원점에 있고 머리는 기하학적 좌표를 가리킴

SECTION 01-2 벡터 연산(1)

1.2.1 두 벡터의 덧셈

- 벡터 합은 동일한 차원을 갖는 벡터끼리만 가능 – 뺄셈도 마찬가지임

식 1-2 두 벡터의 덧셈

$$\begin{bmatrix}4\\5\\6\end{bmatrix}+\begin{bmatrix}10\\20\\30\end{bmatrix}=\begin{bmatrix}14\\25\\36\end{bmatrix}$$

식 1-3 두 벡터의 뺄셈

$$\begin{bmatrix}4\\5\\6\end{bmatrix}-\begin{bmatrix}10\\20\\30\end{bmatrix}=\begin{bmatrix}-6\\-15\\-24\end{bmatrix}$$

- 파이썬에서 두 벡터의 덧셈

```
v = np.array([4,5,6])
w = np.array([10,20,30])
u = np.array([0,3,6,9])
vPlusW = v + w
uPlusW = u + w # 오류! 차원 불일치!
```

SECTION 01-2 벡터 연산(2)

◦ 덧셈에서 벡터 방향

식 1-4 열벡터에 행벡터를 더할 수 있나요?

$$\begin{bmatrix}4\\5\\6\end{bmatrix} + [10 \;\; 20 \;\; 30] = ?$$

- 파이썬의 연산 동작

```
v = np.array([[4,5,6]]) # 행벡터
w = np.array([[10,20,30]]).T # 열벡터
v + w
>> array([[14, 15, 16],
          [24, 25, 26],
          [34, 35, 36]])
```

- 이전에 정의한 벡터 덧셈의 결과와 다름
- 파이썬의 브로드캐스팅(broadcasting) 연산
- 두 벡터의 차원과 방향이 같을 때만 더할 수 있음

SECTION 01-2 벡터 연산(3)

1.2.2 벡터의 덧셈과 뺄셈의 기하학적 구조

- 두 벡터를 기하학적으로 더할 때
  - 한 벡터의 꼬리와 다른 벡터의 머리를 연결
- 두 벡터를 기하학적으로 뺄 때
  - 두 벡터의 꼬리들을 같은 좌표에 위치시킴(기준 위치에 두 벡터를 두면 쉬움)
  - 뺀 결과의 벡터는 두 번째 벡터의 머리에서 첫 번째 벡터의 머리로 가는 선

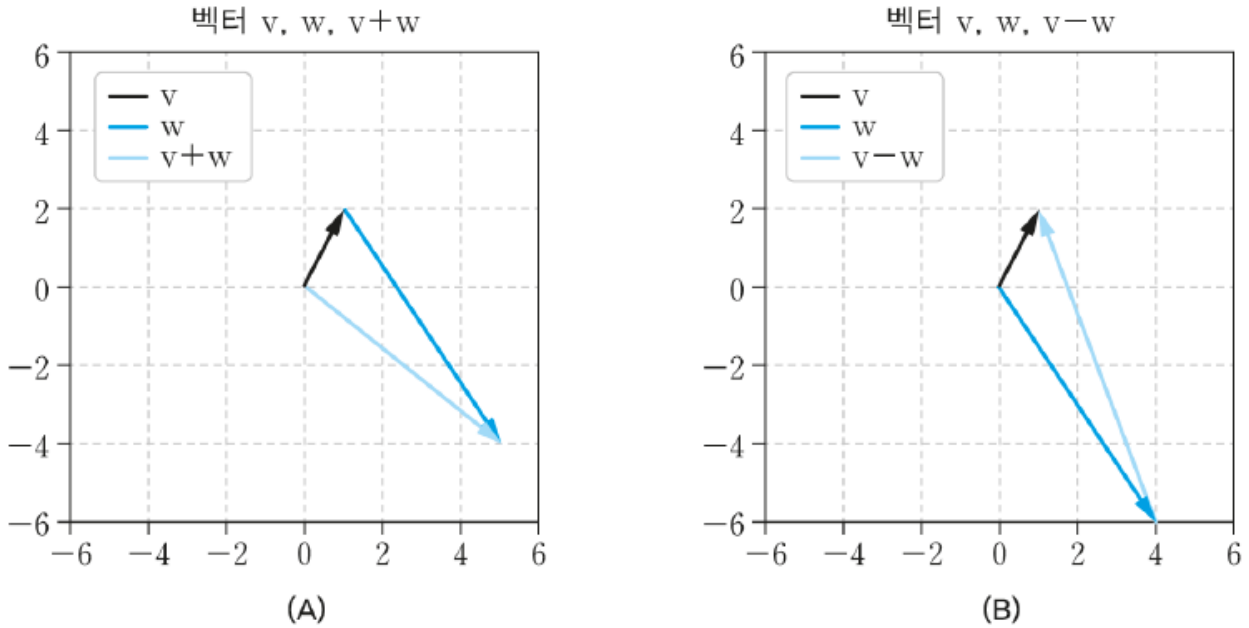


그림 1-2 두 벡터의 합과 차

SECTION 01-2 벡터 연산(4)

1.2.3 스칼라-벡터 곱셈

- 스칼라(scalar): 벡터나 행렬에 포함된 숫자가 아닌 수 그 자체
  - 스칼라는 일반적으로 α또는 λ와 같은 그리스어 소문자로 나타냄
  - 예를 들면 스칼라–벡터 곱셈을 λw로 나타낼 수 있음

식 1-5 스칼라-벡터 곱셈(또는 벡터-스칼라 곱셈)

$$\lambda = 4,\ \mathbf{w} = \begin{bmatrix} 9 \\ 4 \\ 1 \end{bmatrix},\ \lambda\mathbf{w} = \begin{bmatrix} 36 \\ 16 \\ 4 \end{bmatrix}$$

- 스칼라-벡터 곱셈의 데이터 타입

```
s = 2
a = [3,4,5] # 리스트
b = np.array(a) # np 배열
print(a*s)
print(b*s)
```

```
>> [ 3, 4, 5, 3, 4, 5 ]
>> [ 6 8 10 ]
```

- 코드는 스칼라(변수 s)와 벡터 리스트(변수 a)를 생성하고 a를 NumPy 배열(변수 b)로 변환
- 파이썬에서 별(*)표 연산은 변수 타입에 따라 다르게 동작하도록 재정의
- 리스트 반복과 스칼라-벡터 곱셈
- 만약, s=2.0이라면 a*s는 오류가 발생 - 리스트 반복은 정수로만 수행되기 때문임

SECTION 01-2 벡터 연산(5)

1.2.4 스칼라-벡터 덧셈

- 선형대수학에서 벡터와 스칼라는 별도의 수학적 객체이며 결합할 수 없음
- 그러나, 파이썬과 같은 수치 처리 프로그램에서는 벡터에 스칼라를 더할 수 있음
  - 연산은 스칼라–벡터 곱셈과 유사하여, 각 벡터 원소에 스칼라를 더하면 됨

```
s = 2
v = np.array([3,6])
s + v
```

```
>> array([5, 8])
```

SECTION 01-2 벡터 연산(6)

스칼라-벡터 곱셈의 기하학적 구조

- 스칼라는 벡터의 방향을 바꾸지 않고 크기만 조정
- 스칼라–벡터 곱셈의 결과는 스칼라가 1보다 큰지, 0과 1 사이인지, 정확히 0인지, 음수인지에 따라 다름

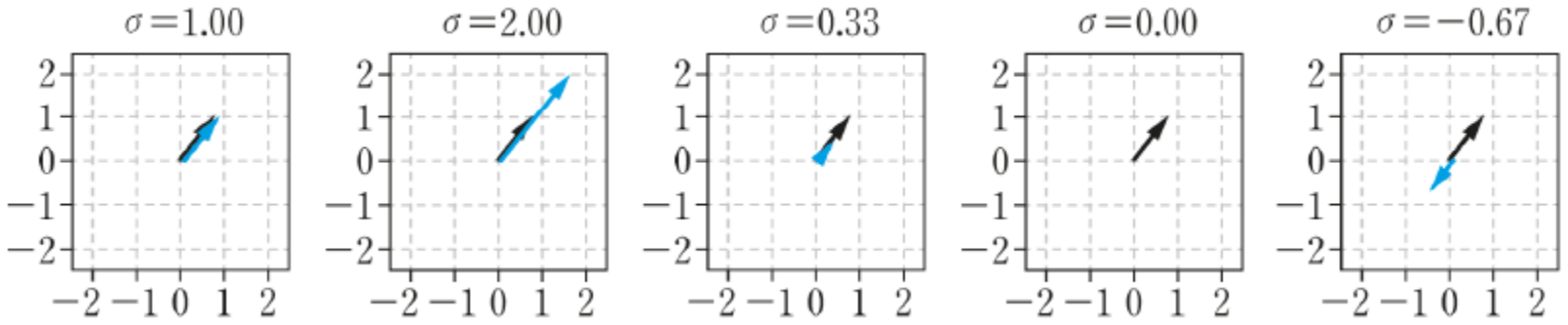


그림 1-3 동일한 벡터(검은 화살표)에 다양한 스칼라 σ(구분하기 위해 회색 선을 살짝 이동)를 곱합니다.

  - 그림에서 스칼라가 음수일 때 벡터 방향이 뒤집힘(즉 180도 회전)
  - '회전된' 벡터는 여전히 동일한 무한한 선을 가리키므로 음의 스칼라가 방향을 바꾼 것이 아님
- 벡터의 평균(vector average) 계산
  - 벡터 합과 스칼라-벡터 곱셈을 이용
  - N개 벡터의 평균을 구하려면 모두 더하고 스칼라 1/N을 곱

SECTION 01-2 벡터 연산(7)

1.2.5 전치

◦ 전치(transpose) 연산: 열벡터를 행벡터로 또는 반대로 변환

- 각 행렬 원소인 (행, 열) 인덱스를 맞바꾸는 것

식 1-6 전치 연산

$$\mathrm{m}^{\mathrm{T}}_{i,j} = \mathrm{m}_{j,i}$$

- 두 번 전치하면 벡터는 원래 방향이 됨
  - 즉, $\mathbf{v}^{TT} = \mathbf{v}$
  - 데이터 과학과 머신러닝 등 여러 중요한 증명의 핵심 근거
  - 예) 데이터 행렬에 자신의 전치를 곱하면 대칭 공분산 행렬이 만들어짐

SECTION 01-2 벡터 연산(8)

1.2.6 파이썬에서 벡터 브로드캐스팅

◦ 브로드캐스팅 연산은 현대 컴퓨터 기반 선형대수학에서만 존재

- 브로드캐스팅은 본질적으로 한 벡터를 다른 벡터의 각 원소로 연산을 여러 번 반복하는 것
- 행렬 합을 브로드캐스팅으로 구현

$$\begin{bmatrix}1 & 1\end{bmatrix}+\begin{bmatrix}10 & 20\end{bmatrix}$$
$$\begin{bmatrix}2 & 2\end{bmatrix}+\begin{bmatrix}10 & 20\end{bmatrix}$$
$$\begin{bmatrix}3 & 3\end{bmatrix}+\begin{bmatrix}10 & 20\end{bmatrix}$$

⇨

```
v = np.array([[1,2,3]]).T # 열벡터
w = np.array([[10,20]]) # 행벡터
v + w # 브로드캐스팅 덧셈
```

```
>> array([[11, 21],
          [12, 22],
          [13, 23]])
```

SECTION 01-3 벡터 크기와 단위벡터(1)

- 벡터의 크기(기하학적 길이 또는 노름(norm)): 벡터의 꼬리부터 머리까지의 거리
  - 표준 유클리드(Euclidean) 거리 공식(벡터 원소들의 제곱합에 제곱근을 취함, [식 1-7] 참고)으로 구함
  - 벡터 크기는 벡터 양 옆에 이중 수직 막대로 표시($\|\mathbf{v}\|$)
    - 일부 응용에서 제곱 크기($\|\mathbf{v}\|^2$)를 사용하는 경우, 오른편에 있는 제곱근 항을 제거

식 1-7 벡터 크기(노름)

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

  - 수학과 파이썬의 용어 차이
    - 수학에서 벡터의 차원은 벡터의 원소 수, 길이는 기하학적 거리
    - 파이썬에서는 함수 len() (len은 length의 약자)은 배열의 차원을 반환하고 np.norm()은 기하학적 길이(크기)를 반환
    - 여기 학습에서는 길이 대신 크기(또는 기하학적 길이)라는 용어를 계속 사용

```
v = np.array([1,2,3,7,8,9])
v_dim = len(v) # 수학의 차원성
v_mag = np.linalg.norm(v) # 수학적 크기, 길이, 또는 노름
```

SECTION 01-3 벡터 크기와 단위벡터(2)

◦ 단위벡터

- 기하학적인 길이가 1인 벡터
- 응용의 예) 직교 행렬과 회전 행렬, 고유벡터, 특이벡터 등
- 단위벡터는 $\| \mathbf{v} \| = 1$ 로 정의

- 연관된 단위벡터를 만들기
  - 벡터 노름의 역수를 스칼라 곱셈

식 1-8 단위벡터 생성

$$\hat{\mathbf{v}} = \frac{1}{\| \mathbf{v} \|} \mathbf{v}$$

- 부모 벡터(v)와 같은 방향의 단위벡터($\hat{v}$)를 표시하는 일반적인 규약

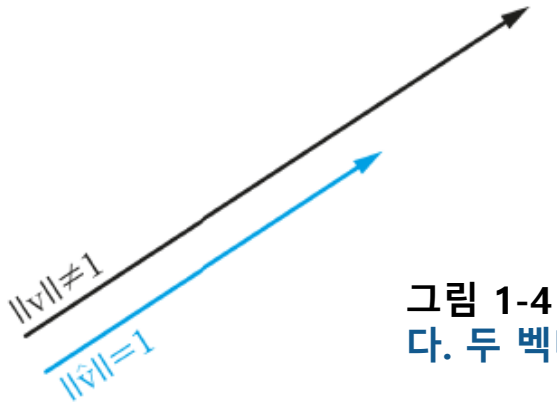


그림 1-4 단위벡터(강조색 화살표)는 비단위벡터(검은 화살표)를 가공해서 만듭니다. 두 벡터의 각도는 동일하지만 크기는 다릅니다.

SECTION 01-4 벡터-내적(1)

- 내적(또는 점곱, 스칼라곱)은 선형대수학 전체에서 가장 중요한 연산
  - 합성곱(convolution), 상관관계(correlation), 푸리에 변환(Fourier transform), 행렬 곱셈, 선형 특징 추출, 신호 필터링 등 많은 연산과 알고리즘의 기본
  - 두 벡터 사이의 내적을 표기하는 방법
    - 일반적인 표기법 $\mathbf{a}^T\mathbf{b}$
    - a·b 또는 $\langle \mathbf{a}, \mathbf{b} \rangle$
  - 내적 계산
    - 두 벡터에서 대응되는 원소끼리 곱한 다음 모든 결과를 더함
    - 내적이 동일한 차원의 두 벡터 사이에서만 성립

식 1-9 내적 공식

$$\delta = \sum_{i=1}^{n} a_i b_i$$

식 1-10 내적 계산의 예

$$\begin{aligned}[1\ 2\ 3\ 4] \cdot [5\ 6\ 7\ 8] &= 1\times5+2\times6+3\times7+4\times8 \\ &= 5+12+21+32 \\ &= 70\end{aligned}$$

SECTION 01-4 벡터-내적(2)

- 파이썬에서 내적을 구현하는 여러 가지 방법 중 np.dot() 함수

```
v = np.array([1,2,3,4])
w = np.array([5,6,7,8])
np.dot(v,w)
```

- 벡터에 스칼라를 곱하면 내적도 그만큼 커짐

```
s = 10
np.dot(s*v,w)
```

  - v와 w의 내적은 70이고, s*v(수학 표기법으로 $\sigma v^T w$)와 w의 내적은 700
  - 음수 스칼라, 예를 들어 s = -1를 시도하면, 내적 크기는 그대로지만 기호는 반대
  - s = 0이면 내적도 0

- 내적은 두 벡터 사이의 유사성(similarity) 또는 매핑(mapping)의 척도
  - 피어슨 상관 계수(Pearson correlation coefficient) - 두 변수 사이의 정규화된 내적

SECTION 01-4 벡터-내적(2)

1.4.1 내적의 분배 법칙

- 벡터 합의 내적은 벡터-내적의 합과 같음

$$a^T(b+c) = a^Tb + a^Tc$$

- 파이썬 코드로 구현한 분배 법칙

```
a = np.array([ 0,1,2 ])
b = np.array([ 3,5,8 ])
c = np.array([ 13,21,34 ])

# 내적 분배 법칙
res1 = np.dot( a, b+c )
res2 = np.dot( a,b ) + np.dot( a,c )
```

- 두 결과인 res1과 res2는 동일(정답은 110)
- 이는 내적의 분배 법칙이 성립함을 나타냄

SECTION 01-4 벡터-내적(3)

1.4.2 내적의 기하학적 해석

- 내적의 기하학적 정의
  - 두 벡터의 크기를 곱하고 두 벡터 사이의 각도에서 코사인값만큼 크기를 증가시킴
  - [식 1-9]와 [식 1-11]은 수학적으로 동일하지만 다르게 표현된 형태

식 1-11 벡터-내적의 기하학적 정의

$$\alpha = \cos(\theta_{\mathbf{v},\mathbf{w}}) \| \mathbf{v} \| \| \mathbf{w} \|$$

- 두 벡터 사이의 각에 따른 내적 부호 다섯 가지 사례

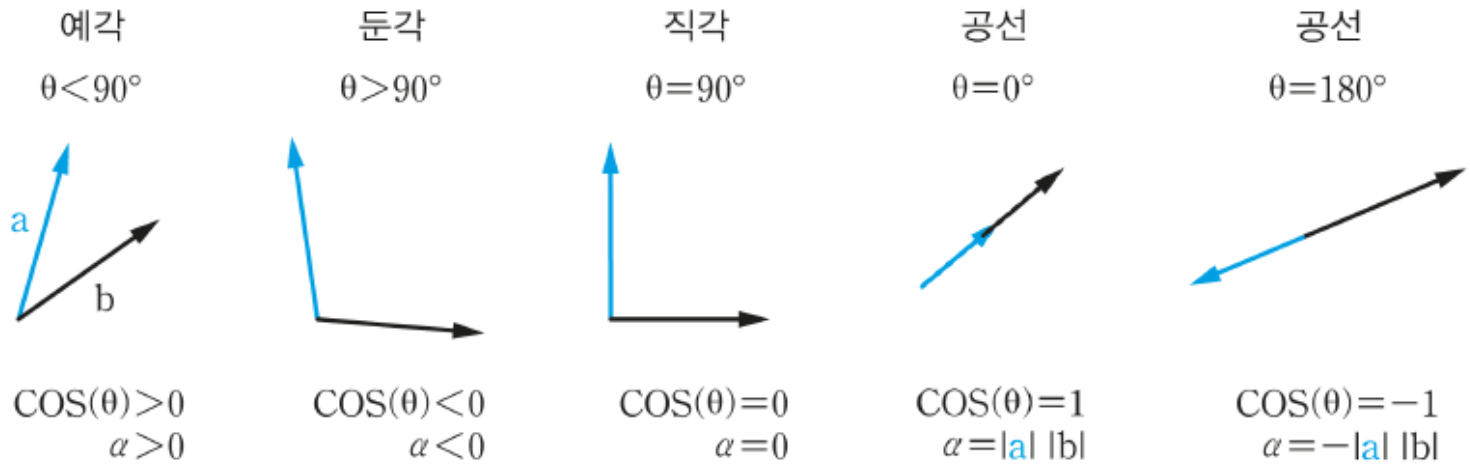


그림 1-5 두 벡터의 내적의 부호는 벡터 사이의 기하학적 관계를 나타냅니다.

SECTION 01-5 그 외 벡터 곱셈(1)

1.5.1 아다마르곱

- 아다마르곱(Hardamard product)의 구현
  - 두 벡터의 대응되는 각 원소를 곱합
  - 곱의 결과는 두 벡터와 같은 차원의 벡터

$$\begin{bmatrix} 5 \\ 4 \\ 8 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ .5 \\ -1 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 4 \\ -2 \end{bmatrix}$$

- 파이썬에서 별표는 두 벡터나 행렬에서 원소 사이의 곱을 나타냄

```
a = np.array([5,4,8,2])
b = np.array([1,0,.5])
a * b
```

- **파이썬에서 위의 코드를 실행하면 오류가 발생**
- **왜 그럴까?**

SECTION 01-5 그 외 벡터 곱셈(2)

1.5.2 외적

- 외적은 열벡터와 행벡터를 이용해 행렬을 생성
  - 외적 행렬의 각 행은 행벡터 스칼라에 대응되는 열벡터 원소를 곱한 것
  - 외적 행렬의 각 열은 열벡터 스칼라에 대응되는 행벡터 원소를 곱한 것

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} [d \ e] = \begin{bmatrix} ad & ae \\ bd & be \\ cd & ce \end{bmatrix}$$

- 외적의 특징
  - 외적은 스칼라 대신 행렬을 생성
  - 외적의 두 벡터는 차원이 달라도 되지만 내적의 두 벡터는 차원이 같아야 함
  - 외적($\mathbf{vw}^T$), 내적($\mathbf{v}^T\mathbf{w}$)
  - 브로드캐스팅은 덧셈, 곱셈, 나눗셈과 같은 산술 연산을 벡터로 확장한 일반적인 코딩 연산
    - 외적은 두 벡터를 곱하는 특수한 수학적 기법
    - NumPy를 이용해 각각 열과 행 방향인 두 벡터를 np.outer() 또는 np.dot() 함수에 입력해서 외적을 계산

SECTION 01-5 그 외 벡터 곱셈(3)

1.5.3 교차곱과 삼중곱

- 교차곱 또는 삼중곱같이 벡터를 곱하는 또 다른 방법도 여럿 있음
- 기하학과 물리학에서 사용되지만 기술 관련 응용 분야에서 자주 등장하지 않아서 이 학습에서는 제외

SECTION 01-6 직교벡터 분해(1)

- 분해 개념
  - 스칼라 분해
    - 숫자 42.01 = 42 + 0.01
    - 소인수 분해(prime factorization) - 숫자 42를 소수 2, 3, 7의 곱으로 분해
  - 벡터 분해
    - 하나의 벡터를 두 개의 벡터로 분해하는데, 하나는 기준 벡터와 직교하고 다른 하나는 기준 벡터와 평행
      - 직교벡터 분해는 통계에서 그람-슈미트 과정(Gram-Schmidt Process)과 QR 분해에 직접적인 연관

SECTION 01-6 직교벡터 분해(2)

- 벡터 분해 시각화
  - 학습 예시
    - 표준 위치에 두 개의 벡터 **a** 와 **b** 가 존재
    - **a**에서 **b** 의 머리와 최대한 가까운 점을 탐색
      - 최적화 문제로 표현 가능. 즉 투영 거리가 최소가 되도록 벡터 **b** 를 벡터 **a** 에 투영
      - 그 점은 **a** 의 크기를 줄인 즉 βa
      - 스칼라 β를 찾으면 됨

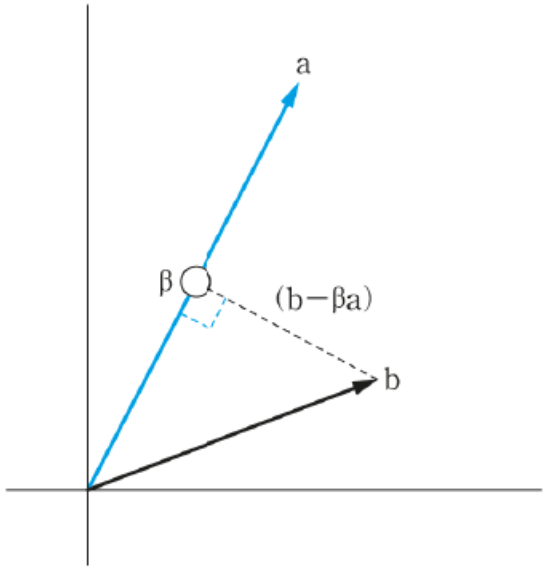


그림 1-6 b 의 머리와 가장 가까운 벡터 a 위의 점을 찾으려면 투영 벡터 b- βa 의 길이가 최소가 되는 β를 구하는 공식이 필요합니다.

SECTION 01-6 직교벡터 분해(3)

- 직교 투영법
  - 앞의 **b**-β**a**가 β**a**와 직교한다는 것을 추론 가능
    - 즉 이 벡터들은 수직. 따라서, 둘 사이의 내적이 0이 되어야 함

$$\mathbf{a}^{\mathrm{T}}(\mathbf{b}-\beta\mathbf{a})=0$$

  - β 구하기

식 1-12 직교 투영 문제를 풉니다.

$$\begin{aligned}\mathbf{a}^{\mathrm{T}}\mathbf{b}-\beta\mathbf{a}^{\mathrm{T}}\mathbf{a}&=0\\ \beta\mathbf{a}^{\mathrm{T}}\mathbf{a}&=\mathbf{a}^{\mathrm{T}}\mathbf{b}\\ \beta&=\frac{\mathbf{a}^{\mathrm{T}}\mathbf{b}}{\mathbf{a}^{\mathrm{T}}\mathbf{a}}\end{aligned}$$

- '목표 벡터'와 '기준 벡터'
  - 목적은 목표 벡터를 두 개의 다른 벡터로 분해
    - (1) 그 두 벡터의 합은 목표 벡터
    - (2) 하나의 벡터는 기준 벡터와 직교하지만 다른 벡터는 기준 벡터와 평행

SECTION 01-6 직교벡터 분해(4)

- 용어 정리
  - 목표 벡터 **t**, 기준 벡터 **r**
  - 표 벡터로부터 만들어진 두 벡터는 수직 성분 $\mathbf{t}_{\perp r}$과 평행 성분 $\mathbf{t}_{\parallel r}$

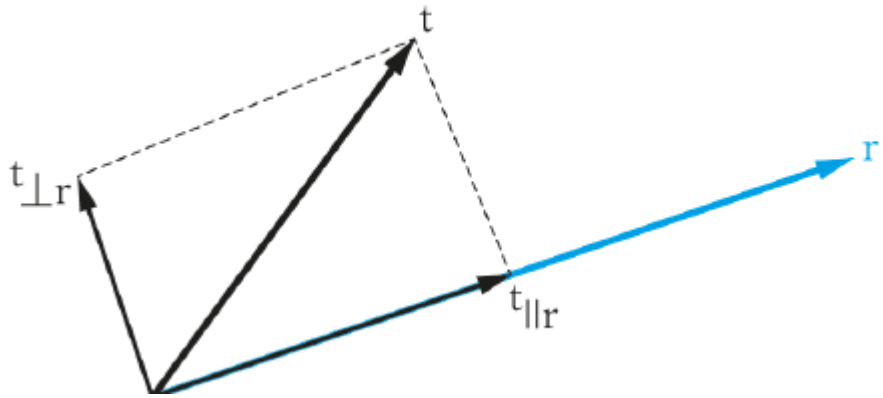


그림 1-7 직교벡터 분해 그림: 벡터 t를 벡터 r과 직교하는 벡터와 평행한 벡터로 분해합니다

SECTION 01-6 직교벡터 분해(5)

- 평행 성분
  - r 의 크기를 조정한 벡터는 r과 평행
  - [식 1-12]에서는 스칼라 β만 계산. 여기서는 크기를 조절한 벡터 β을 계산
  - 두 벡터 성분의 합이 목표 벡터

$$t = t_{\perp r} + t_{\parallel r}$$

$$t_{\perp r} = t - t_{\parallel r}$$

SECTION 01-6 직교벡터 분해(6)

- 수직 성분
  - 수직 성분이 정말로 기준 벡터와 직교할까?
  - (증명) 수직 성분과 기준 벡터 사이의 내적이 0인지 계산

$$(\mathbf{t}_{\perp \mathbf{r}})^{\mathrm{T}}\mathbf{r} = 0$$

$$\left(\mathbf{t} - \mathbf{r}\frac{\mathbf{t}^{\mathrm{T}}\mathbf{r}}{\mathbf{r}^{\mathrm{T}}\mathbf{r}}\right)^{\mathrm{T}}\mathbf{r} = 0$$

SECTION 01-7 정리

- 벡터는 열 또는 행에 숫자를 나열한 것
  - 벡터의 원소 수를 차원이라고 하며, 벡터는 차원과 동일한 수의 축을 가진 기하학적 공간에서 하나의 선으로 나타낼 수 있음
- 덧셈, 뺄셈, 아다마르곱과 같은 벡터 산술 연산은 원소별로 계산
- 내적은 차원이 같은 두 벡터 간의 관계를 인코딩한 단일 숫자로, 원소별로 곱하고 합해서 구함
- 두 벡터가 직교하면 내적은 0이며 기하학적으로 벡터가 직각으로 만나는 것을 의미
- 직교벡터 분해는 하나의 벡터를 기준 벡터와 직교하는 벡터, 평행한 벡터로 나누는 것
- 분해 공식은 기하학적으로 도출될 수 있지만, 공식이 내포한 개념인 '크기에 대한 매핑'이라는 문구를 기억

SECTION 01 연습 문제(1)

[연습 문제 1-1]

- [그림 1-2]를 생성하는 코드를 스스로 작성

[연습 문제 1-2]

- [식 1-7]을 코드로 변환해서 벡터의 노름을 계산하는 알고리즘을 작성

[연습 문제 1-3]

- 벡터를 입력으로 받아 동일한 방향의 단위벡터를 출력하는 파이썬 함수를 구현

[연습 문제 1-4]

- 벡터와 원하는 크기를 입력 받고 벡터와 동일한 방향이지만 두 번째 입력에 해당하는 크기를 가진 벡터를 반환하는 파이썬 함수를 작성

[연습 문제 1-5]

- np.transpose(v) 또는 v.T와 같은 내장 함수 또는 메서드를 사용하지 않고 행벡터를 열벡터로 전치하는 for 루프를 작성

[연습 문제 1-6]

- 벡터의 제곱 노름을 그 벡터 자체의 내적으로 계산

SECTION 01 연습 문제(2)

[연습 문제 1-7]

- 내적의 교환법칙을 입증하는 코드를 작성

[연습 문제 1-8]

- 코드로 [그림 1-6]을 만들기

[연습 문제 1-9]

- 직교벡터 분해를 구현. 두 난수 벡터 t와 r로 시작해서 [그림 1-8]을 재현

[연습 문제 1-10]

- 코드 버그로 인해 [식 1-13]의 투영 스칼라의 분모가 r^Tr이 아닌 t^Tt이 되었다고 가정(이 장을 쓰는 동안 직접 경험한 발생하기 쉬운 실수). 이 버그를 구현해서 정확한 코드와 얼마나 차이가 있는지 확인
- 온전성 검사(sanity-checking)

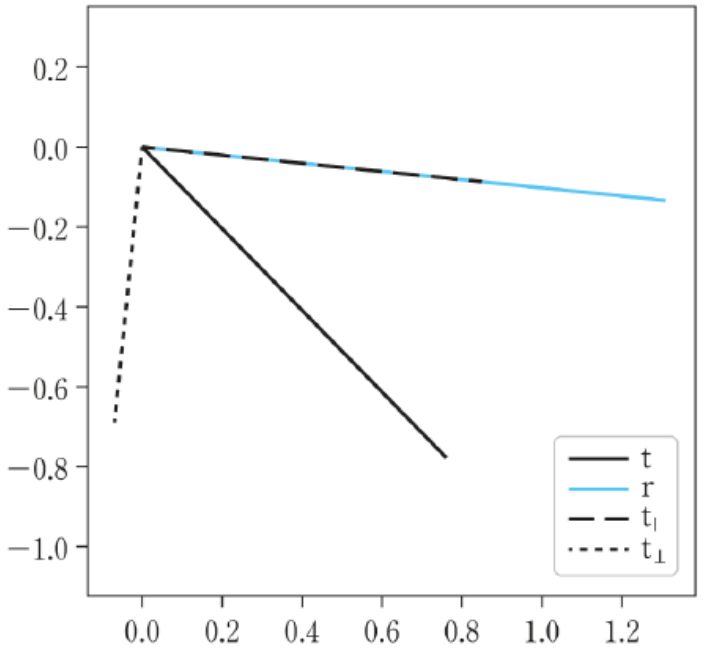


그림 1-8 [연습 문제 1-9]의 결과

CHAPTER 02: 벡터, 파트 2: 벡터의 확장 개념

Contents

- CHAPTER 02: 벡터, 파트 2: 벡터의 확장 개념

SECTION 02-1 벡터 집합

- 벡터 집합(set): 벡터들의 모음
  - 벡터 집합은 S 또는 V와 같이 대문자 이탤릭체로 표시
  - 수학적 표현

$$V = \{\mathbf{v}_1, \cdots, \mathbf{v}_n\}$$

  - 벡터 집합은 유한 또는 무한한 수의 벡터를 가질 수 있음
  - 벡터 집합이 비어 있다면 $V = \{\ \}$

SECTION 02-2 선형 가중 결합(1)

- 선형 가중 결합(linear weighted combination)
  - 여러 변수마다 가중치를 다르게 주어 정보를 혼합하는 방법
  - 선형 혼합물(linear mixture) 또는 가중 결합(weighted combination)(선형이라고 가정)
  - 계수(coefficient)
  - 스칼라-벡터 곱을 한 다음 합하기

식 2-1 선형 가중 결합

$$\mathbf{w} = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \ldots + \lambda_n \mathbf{v}_n$$

식 2-2 선형 가중 결합

$$\lambda_1 = 1,\ \lambda_2 = 2,\ \lambda_3 = -3, \quad \mathbf{v}_1 = \begin{bmatrix} 4 \\ 5 \\ 1 \end{bmatrix},\ \mathbf{v}_2 = \begin{bmatrix} -4 \\ 0 \\ -4 \end{bmatrix},\ \mathbf{v}_3 = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

$$\mathbf{w} = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \lambda_3 \mathbf{v}_3 = \begin{bmatrix} -7 \\ -4 \\ -13 \end{bmatrix}$$

SECTION 02-2 선형 가중 결합(2)

- 선형 가중 결합의 코드 구현

```
l1 = 1
l2 = 2
l3 = -3
v1 = np.array([4,5,1])
v2 = np.array([-4,0,-4])
v3 = np.array([1,3,2])
l1*v1 + l2*v2 + l3*v3
```

- 선형 가중 결합의 응용
  - 통계 모델로부터 예측된 데이터는 최소제곱 알고리즘을 통해 계산되는 회귀 변수(regressor, 독립변수)와 계수(스칼라)의 선형 가중 결합으로 생성
  - 주성분 분석과 같은 차원 축소 과정에서 각 성분(인자 또는 모드)은 성분의 분산을 최대화하는 가중치(계수)와 데이터 채널의 선형 가중 결합으로 도출
  - 인공 신경망(딥러닝의 기반 아키텍처 및 알고리즘)의 두 가지 연산, 즉 입력 데이터의 선형 가중 결합과 비선형 변환

SECTION 02-3 선형 독립성(1)

- 선형 종속적(linearly dependent)
  - 벡터 집합에서 적어도 하나의 벡터를 집합의 다른 벡터들의 선형 가중 결합으로 나타낼 수 있을 때 벡터 집합
- 선형 독립적(linearly independent)
  - 집합에 있는 벡터들의 선형 가중 결합으로 집합의 아무런 벡터도 나타낼 수 없을 때

$$V = \left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 7 \end{bmatrix} \right\}$$

- **벡터 집합 V 는 선형 독립적**
- **집합의 한 벡터를 집합의 다른 벡터의 선형 배수로 나타낼 수 없음**
  즉, 집합 내의 벡터들을 v_1과 v_2 라고 했을 때 $v_1 = \lambda v_2$인 스칼라 λ가 존재하지 않음

$$S = \left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 6 \end{bmatrix} \right\}$$

- **벡터 집합 S는 선형 종속적**
- **집합의 벡터를 선형 가중 결합해서 집합의 다른 벡터를 만들 수 있기 때문임**
- **이러한 결합은 무한히 존재하며 그중 두 가지는 $S_1 = .5*S_2$와 $S_2 = 2*S_1$**

SECTION 02-3 선형 독립성(2)

- 선형 독립성을 결정하는 방법
  - 벡터 집합으로 행렬을 만들고 행렬의 계수를 계산한 다음 행의 수와 열의 수 중에서 더 작은 값과 비교

$$T = \left\{ \begin{bmatrix} 8 \\ -4 \\ 14 \\ 6 \end{bmatrix}, \begin{bmatrix} 4 \\ 6 \\ 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 14 \\ 2 \\ 4 \\ 7 \end{bmatrix}, \begin{bmatrix} 13 \\ 2 \\ 9 \\ 8 \end{bmatrix} \right\}$$

SECTION 02-3 선형 독립성(3)

2.3.1 수학에서의 선형 독립성

- 선형 종속의 수학적 정의
  - 선형 종속적이라면 집합의 벡터들의 선형 가중 결합으로 영벡터를 만들수 있음
  - 식을 0과 같다고 설정하면 전체 집합이 종속적 또는 독립적
    - 어떤 개별 벡터도 '종속 벡터'라는 특권을 가지지 않게 됨
      다시 말해 독립에 관해서 평등한 벡터 집합

식 2-3 선형 종속성

$$0 = \lambda_1 \mathbf{v}_1 + \lambda_2 \mathbf{v}_2 + \ldots + \lambda_n \mathbf{v}_n, \quad \lambda \in \mathbb{R}$$

  - 적어도 하나의 λ≠0이라는 제약 조건

$$0 = \mathbf{v}_1 + \ldots + \frac{\lambda_n}{\lambda_1} \mathbf{v}_n, \quad \lambda \in \mathbb{R}, \ \lambda_1 \neq 0$$

SECTION 02-3 선형 독립성(4)

2.3.2 독립성과 영벡터

- 영벡터에 스칼라를 곱하면 여전히 영벡터
  - λ≠0이며 자명하지 않은 해법(nontrivial solution)이 존재한다면 그 집합은 선형 종속성 정의에 부합

$$\lambda_0 0 = 0\mathbf{v}_1 + 0\mathbf{v}_2 + 0\mathbf{v}_n$$

SECTION 02-4 부분공간과 생성(1)

2.3.2 독립성과 영벡터

- 집합 안에서 벡터들을 선형 결합할 수 있는 무한한 방법
  - (유한한) 벡터 집합의 동일한 벡터들을 사용하지만 다른 가중치 숫자를 사용해서 무한히 선형 결합하는 방식으로 벡터 부분공간을 생성
  - 벡터 집합의 생성(span): 가능한 모든 선형 가중 결합을 구성하는 메커니즘

$$V = \left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix} \right\}$$

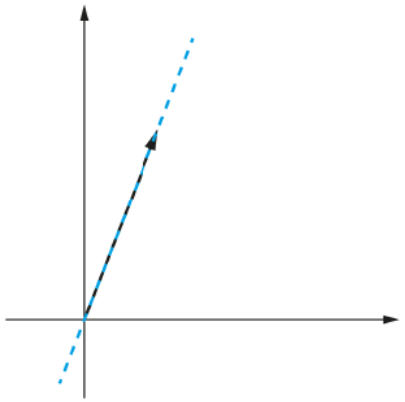


그림 2-1 벡터(검은색)와 벡터가 생성하는 부분공간(강조색)

$$V = \left\{ \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix} \right\}$$

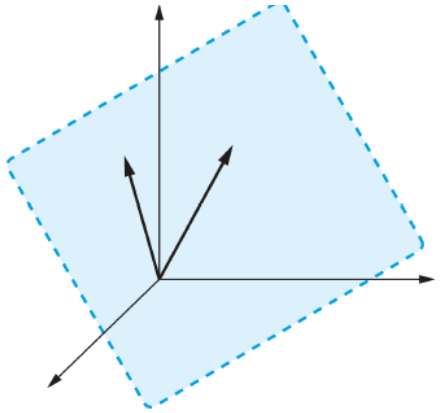


그림 2-2 두 벡터(검은색)와 이들이 생성하는 부분공간(강조색)

$$V = \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} \right\}$$

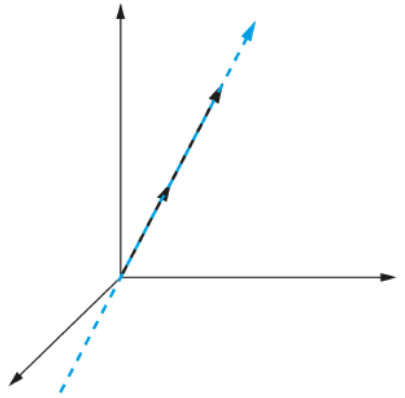


그림 2-3 2개의 벡터(검은색)로 생성된 1차원 부분공간(강조색)

SECTION 02-4 부분공간과 생성(2)

- 생성 부분공간의 차원과 집합의 벡터 수 사이의 관계
  - 벡터 집합에서 생성되는 부분공간의 차원은 선형 독립 집합을 형성하는 데 필요한 최소한의 벡터 수
  - 벡터 집합이 선형 독립적이면 해당 집합의 벡터들로 생성된 부분공간의 차원은 집합의 벡터 수와 동일
  - 반대로 종속적이라면 해당 벡터들로 생성된 부분공간의 차원은 반드시 해당 집합의 벡터 수보다 작음
  - 벡터 부분공간의 공식적인 정의는 덧셈과 스칼라 곱셈으로 닫혀 있는 부분집합으로 공간의 원점을 포함
    - 모든 가중치를 0으로 설정해서 만들어진 공간의 원점인 영벡터도 마찬가지

SECTION 02-5 기저(1)

- 기저(basis)는 일종의 공간을 측정하기 위한 자
  - 기저 단위를 부여해야 의미가 존재
    - 암스테르담(네덜란드)과 테네리페(스페인) 사이의 거리는 대략 2,000 **마일**
  - 기저는 행렬의 정보(예, 데이터)를 설명하는 데 사용하는 자(ruler)의 집합
    - 데카르트 좌표계(Cartesian Coordinate System)
    - 데카르트 기저 집합은 서로 직교하며 단위 길이인 벡터로 구성
      - 표준 기저 집합(standard basis set)

$$S_2 = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \quad S_3 = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

2차원과 3차원 데카르트 그래프의 기저 집합

$$T = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -3 \\ 1 \end{bmatrix} \right\}$$

$\mathbb{R}^2$ 의 또 다른 기저 집합

SECTION 02-5 기저(2)

- 기저 집합 S_2 와 T는 모두 동일한 부분공간을 생성($\mathbb{R}^2$ 전체).
  - S보다 T를 선호
  - 그림 2-4의 데이터 점 p와 q는 기저 집합에 관계없이 동일하지만 T는 간결하고 직관적

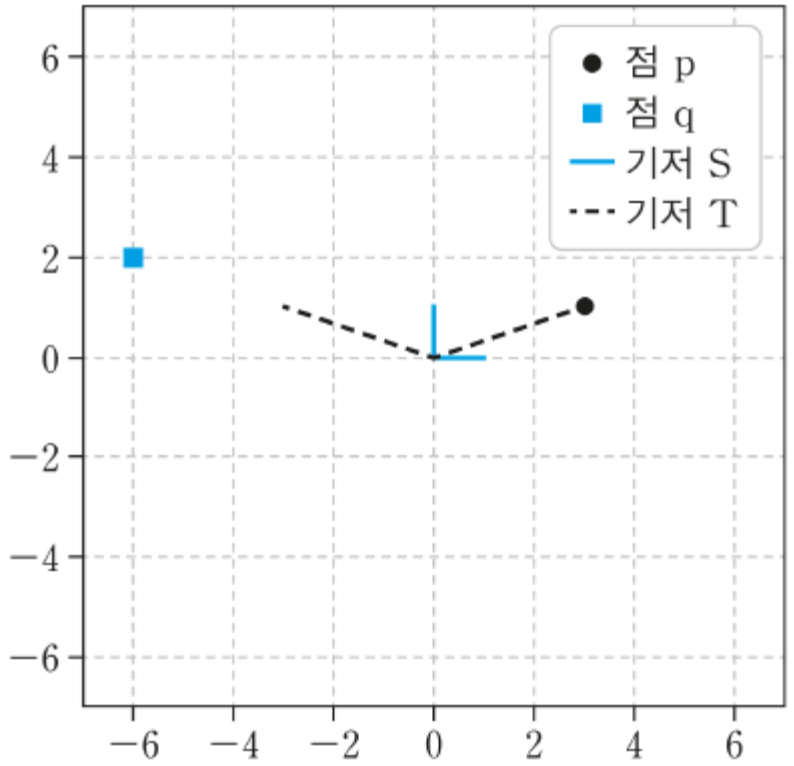


그림 2-4 동일한 점(p와 q)을 기저 집합 S(강조색 실선) 또는 T(검은색 점선)로 기술

SECTION 02-5 기저(3)

- 그림 2-5는 두 변수를 가진 데이터 집합(각 점은 데이터)
  - 세 개의 다른 기저 집합 중 최적의 기저 집합을 모색 – 분석 목표, 데이터 특성, 분석의 제약 조건 등을 고려
    - x = 0과 y = 0 줄에 해당하는 '표준 기저 집합'
    - 주성분 분석(PCA, 왼쪽 도표)
    - 독립 성분 분석(ICA, 오른쪽 도표)

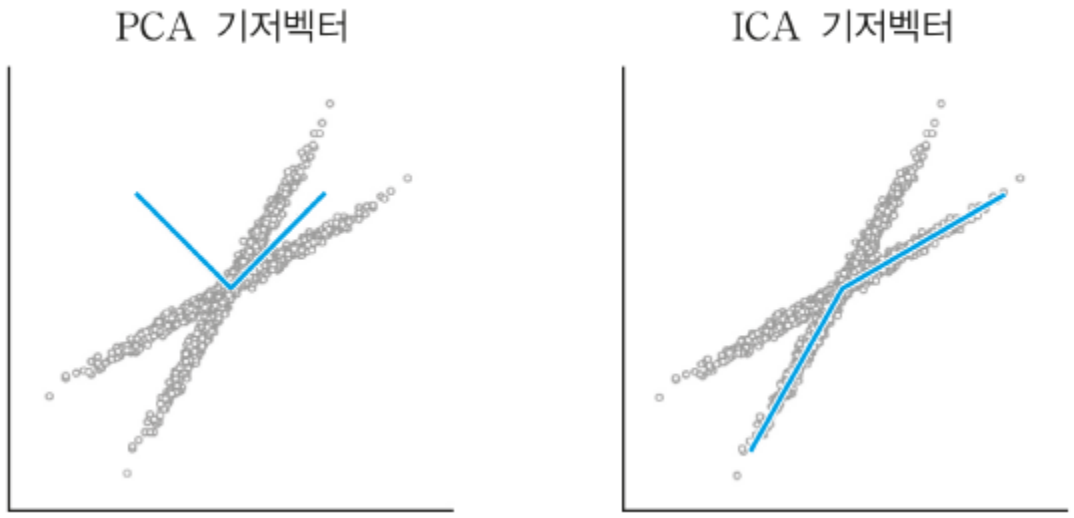


그림 2-5 다른 기저벡터(강조색 선)를 사용한 2차원 데이터 집합

SECTION 02-5 기저(4)

2.5.1 기저 정의

- 기저는 단순히 생성과 독립성을 결합한 것
  - 벡터 집합이 (1)특정 부분공간을 생성하고 (2)독립적인 벡터 집합이라면 해당 부분공간의 기저

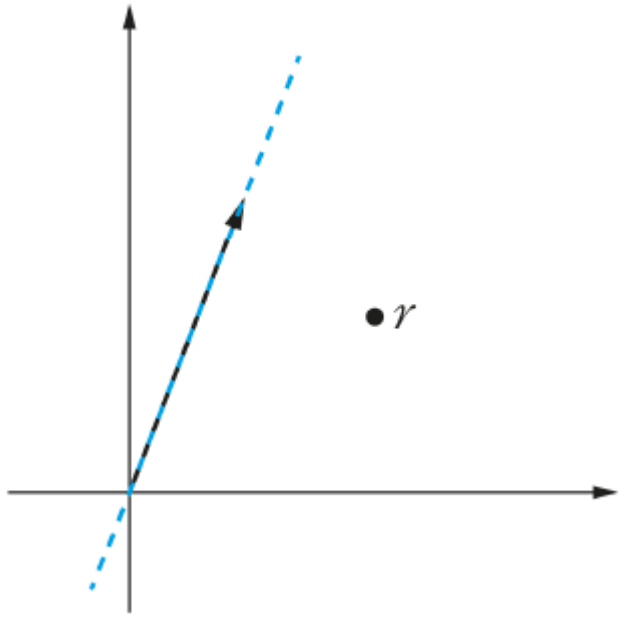


그림 2-6 기저 집합은 생성 내에 포함된 것만 측정할 수 있습니다.

SECTION 02-5 기저(5)

- 기저 집합은 선형 독립이어야 하는 이유
  - 부분공간에 있는 모든 벡터는 그 기저를 이용한 고유한 좌표를 가져야 하기 때문임
  - 모든 벡터가 기저 집합 내에서 고유한 좌표를 가져야 함
    - 고유성을 보장하려면 선형 독립적이 필요

$$U = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

- **U는 완벽하게 유효한 벡터 집합이지만 기저 집합은 절대 아님**

- 하나의 기저 집합 내에서 하나의 점은 정확히 하나의 선형 가중 결합으로 정의
  - 측정 단위마다 하나의 값만 존재
  - 3,200마일과 2,000마일이 동시에 가능한 것이 아니라 3,200 킬로미터와 2,000 마일이 동시에 가능

SECTION 02-6 정리

- 벡터 집합은 벡터들의 모음
  - 집합에는 유한하거나 무한한 수의 벡터가 존재
- 선형 가중 결합은 집합의 벡터들에 스칼라를 곱하고 합하는 것
  - 선형 가중 결합은 선형대수학에서 가장 중요한 개념
- 집합에서 하나의 벡터가 집합의 다른 벡터들의 선형 가중 결합으로 기술될 수 있으면 해당 벡터 집합은 선형 종속적
  - 만약 이러한 선형 가중 결합이 없다면 집합은 선형 독립적
- 부분공간은 벡터 집합의 가능한 모든 선형 가중 결합으로 만들어진 무한 집합
- 기저는 공간을 측정하기 위한 일종의 자
  - 벡터 집합이 (1)어떤 부분공간을 생성하고 (2)선형 독립적이라면 해당 부분공간에 대한 기저가 됨
  - 데이터 과학의 주요 목표는 데이터 집합을 설명하거나 문제를 해결하기 위한 최상의 기저 집합을 찾는 것

SECTION 02 연습 문제(1)

[연습 문제 2-1]

- 선형 가중 결합에 대한 코드를 다시 구현하고 같은 결과가 나오는지 확인

[연습 문제 2-2]

- 리스트의 원소로 스칼라와 벡터를 추가해 보면서 확인

[연습 문제 2-3]

- 기저벡터에 무작위 스칼라를 곱해서 부분공간에 점 100개를 만들고 그 점들을 그리기
- $\mathbb{R}^3$의 [3, 5, 1]과 [0, 2, 2] 두 벡터를 사용해 똑같이 이 과정을 반복

```
import plotly.graph_objects as go
fig = go.Figure( data=[go.Scatter3d(
                    x=points[:,0], y=points[:,1], z=points[:,2],
                    mode='markers' )])
fig.show()
```

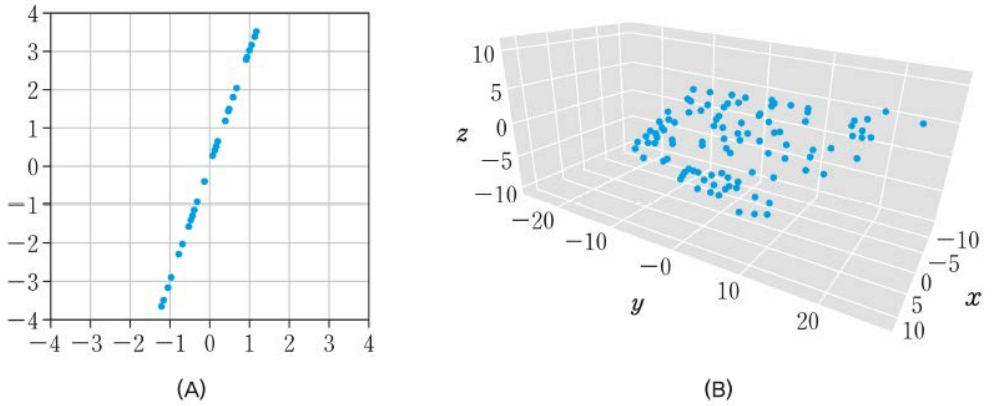


그림 2-7 [연습 문제 2-3]의 결과

CHAPTER 03: 벡터 응용: 데이터 분석에서의 벡터

Contents

- CHAPTER 03: 벡터 응용: 데이터 분석에서의 벡터

SECTION 03-1 상관관계와 코사인 유사도
SECTION 03-2 시계열 필터링과 특징 탐지
SECTION 03-3 k-평균 클러스터링

SECTION 03-1 상관관계와 코사인 유사도(1)

- 상관계수(correlation coefficient)
  - 두 변수 사이의 선형 관계를 정량화한 하나의 숫자
  - 상관계수의 범위는 −1부터 +1까지
    - −1은 완벽한 음의 관계, +1은 완벽한 양의 관계, 0은 선형 관계가 없음

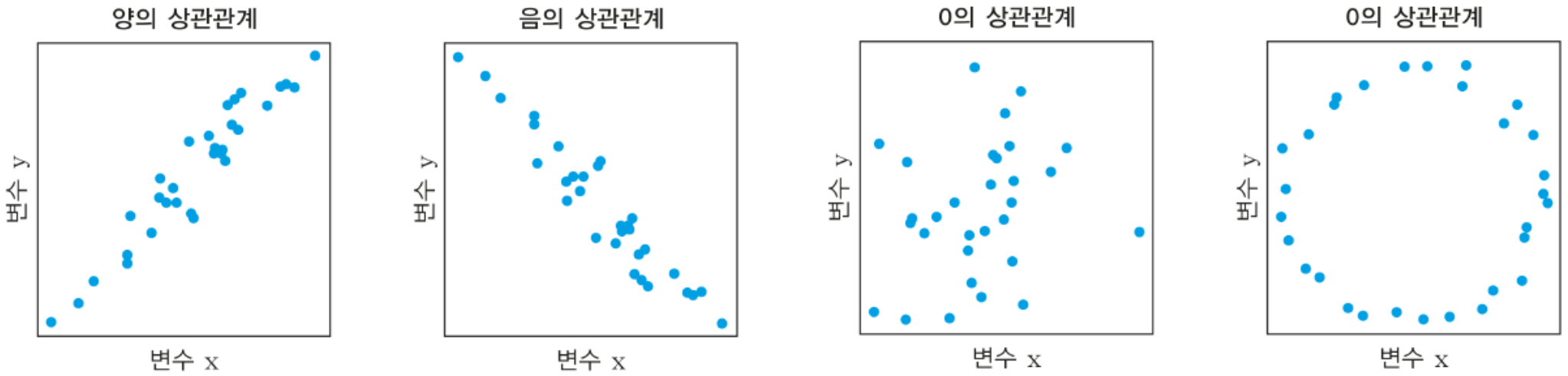


그림 3-1 양의 상관관계, 음의 상관관계, 0의 상관관계를 나타낸 데이터의 예제. 맨 오른쪽 그림은 상관관계가 선형적 척도라는 것을 나타냅니다(상관계수가 0이라도 변수 간의 비선형 관계는 존재할 수 있습니다).

SECTION 03-1 상관관계와 코사인 유사도(2)

◦ 상관계수가 기대하는 범위 −1과 +1 사이에 존재하려면 정규화가 필요

- 각 변수의 평균중심화
  - 평균중심화는 각 데이터값에서 평균값을 빼는 것
- 벡터 노름 곱으로 내적을 나누기
  - 분할(divisive) 정규화는 측정 단위를 제거하고 상관계수 최대 크기를 |1|로 조정

식 3-1 피어슨 상관계수 공식

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

식 3-2 선형대수학 용어로 나타낸 피어슨 상관계수

$$\rho = \frac{\tilde{\mathbf{x}}^{\mathrm{T}}\tilde{\mathbf{y}}}{\|\tilde{\mathbf{x}}\| \quad \|\tilde{\mathbf{y}}\|}$$

- 코사인 유사도(cosine similarity)
  - 코사인 유사도에 대한 공식은 단순히 내적의 기하학적 공식([식 1-11])으로 코사인 항을 구한 것
  - α는 **x**와 **y**의 내적

$$\cos(\theta_{x,y}) = \frac{\alpha}{\|\mathbf{x}\| \quad \|\mathbf{y}\|}$$

SECTION 03-2 시계열 필터링과 특징 탐지

- 내적은 시계열 필터링에도 사용
  - 커널과 시계열 신호 사이의 내적을 계산하는 것이 필터링 메커니즘
  - 필터링할 때 일반적으로 지역(local) 특징 탐지를 해야 하고 커널은 일반적으로 전체 시계열보다 훨씬 짧음
    - 커널과 동일한 길이의 짧은 데이터 조각과 커널 사이의 내적을 계산
    - 이 과정으로 필터링된 신호([그림 3-2]) 구간에서 한 점이 생성되고 커널을 오른쪽으로 한 구간씩 이동시키면서 다른(또는 겹쳐진) 신호 조각과 내적을 계산 - 합성곱(convolution)

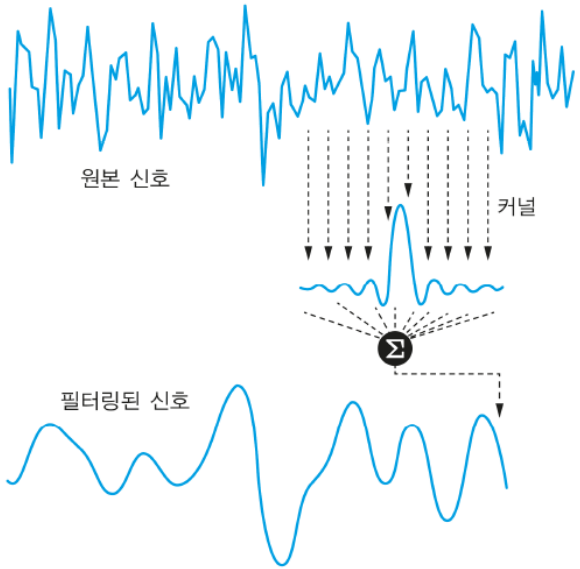


그림 3-2 시계열 필터링 도식화

SECTION 03-3 *k*-평균 클러스터링(1)

- *k*-평균 클러스터링(k-means clustering)
  - 그룹 중심까지의 거리를 최소화하도록 다변량 데이터를 상대적으로 적은 수의 그룹 또는 범주로 분류하는 비지도 기법
  - *k*-평균 클러스터링 구현할 알고리즘
    1. 데이터 공간에서 임의의 *k*개 중심 점을 초기화. 여기서 중심은 클래스 또는 범주이며, 다음 단계에서는 각 데이터 관측치를 각 클래스에 할당
       (중심은 임의의 차원의 수로 일반화된 형태입니다).
    2. 각 데이터 관측치와 각 중심 사이의 유클리드 거리를 계산
    3. 각 데이터 관측치를 가장 가까운 중심의 그룹에 할당
    4. 각 중심을 해당 중심에 할당된 모든 데이터 관측치의 평균으로 갱신
    5. 수렴 기준을 만족할 때까지 또는 *N* 회까지 2~4단계를 반복

SECTION 03-3 *k*-평균 클러스터링(2)

[1 단계] - 무작위로 클러스터 중심 *k*개를 초기화

- *k*는 *k*-평균 클러스터링의 매개변수적 (여기서는 *k* = 3으로 고정)
- *k* 개의 데이터 샘플을 무작위로 선택
- 데이터는 data 변수(150개의 관측치와 2개의 특징으로 150 × 2 모양)에 저장하고 시각화

```
k = 3
ridx = np.random.choice(range(len(data)),k,replace=False)
centroids = data[ridx,:] # data 행렬은 특징별 샘플입니다.
```

SECTION 03-3 *k*-평균 클러스터링(3)

[2 단계] - 각 데이터 관측치와 각 클러스터 중심 사이의 거리를 계산

- 하나의 데이터 관측치와 중심 사이의 유클리드 거리 계산식

$$\delta_{i,j} = \sqrt{(d_i^x - c_j^x)^2 + (d_i^y - c_j^y)^2}$$

- δ_{ij}는 데이터 관측치 i에서 중심 j까지의 거리
- d_i^x는 데이터 관측치 i의 특징 x
- c_j^x는 중심 j의 x축 좌표

- 벡터와 브로드캐스팅으로 작성한 코드

```
dists = np.zeros((data.shape[0],k))
for ci in range(k):
  dists[:,ci] = np.sum((data-centroids[ci,:])**2,axis=1)
```

- 파이썬의 브로드캐스팅은 클러스터 중심을 150회 반복해서 각 데이터 관측치마다 중심을 뺌
- 지수 연산 **은 원소별로 적용되며 axis=1은 파이썬이 열에 걸쳐서(행마다 분리해서) 합을 구하도록 명령을 내림
- 따라서 np.sum()의 출력은 중심 ci에 대한 각 점의 유클리드 거리를 인코딩하는 150 × 1 배열이 됨

유클리드 거리 공식의 제곱근이 코드에는 빠져 있음
그러면 코드가 잘못된 것일까?

SECTION 03-3 *k*-평균 클러스터링(4)

[3 단계] - 각 데이터 관측치를 가장 가까운 거리의 그룹에 할당

- 파이썬으로 구현한 코드

```
groupidx = np.argmin(dists,axis=1)
```

  - np.min - 최솟값을 반환
  - np.argmin - 최솟값의 인덱스를 반환

[4 단계] - 클래스 내의 모든 데이터 점의 평균을 계산해서 중심을 다시 설정

- *k* 개의 클러스터에 루프를 쓰면서 파이썬 인덱싱을 사용하여 각 클러스터에 할당된 모든 데이터 포인트를 탐색

```
for ki in range(k):
centroids[ki,:] = [ np.mean(data[groupidx==ki,0]),
                    np.mean(data[groupidx==ki,1]) ]
```

[5 단계] - 좋은 답을 얻을 때까지 이전 단계들을 반복

SECTION 03-3 *k*-평균 클러스터링(5)

[그림 3-3]의 4개 그림은 초기 무작위 클러스터 중심(반복 0)과 3회 반복 후 갱신된 위치

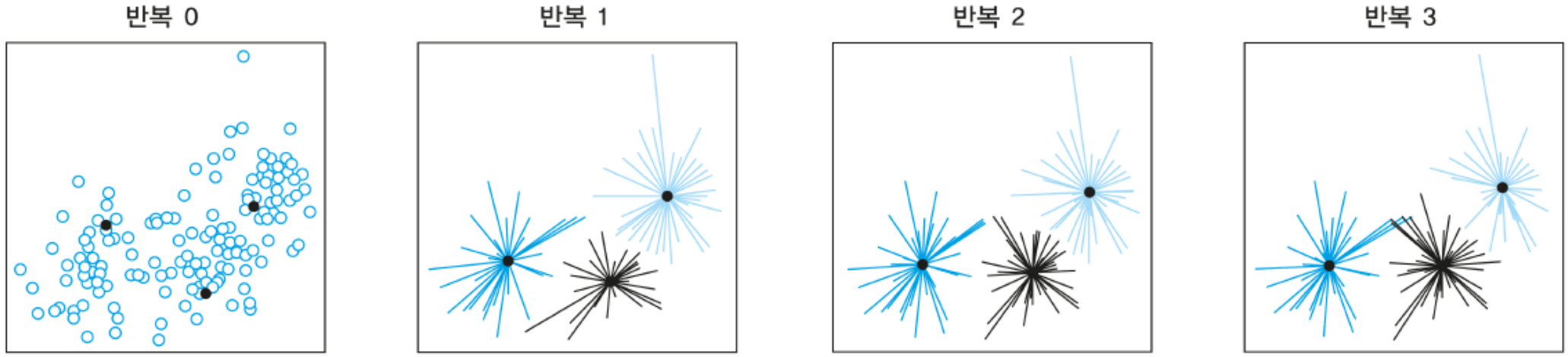


그림 3-3 k-평균

SECTION 03 연습 문제(1)

3.4.1 상관관계 연습 문제

[연습 문제 3-1]

- 두 벡터를 입력으로 받아 두 개의 수를 출력하는 파이썬 함수를 작성

[연습 문제 3-2]

- 상관관계와 코사인 유사도 사이의 차이를 확인하고 그래프 작성

[연습 문제 3-3]

- pearsonr 함수로 피어슨 상관계수를 계산

[연습 문제 3-4]

- 상관관계 함수가 NumPy의 corrcoef 함수보다 빠른지 테스트

SECTION 03 연습 문제(2)

3.4.2 필터링과 특징 탐지 연습 문제

[연습 문제 3-5]

- 에지 검출기 구현

[연습 문제 3-6]

- 다른 신호와 커널을 사용해서 동일하게 과정을 수행
- 목표: 울퉁불퉁한 시계열을 매끄럽게 만들기

[연습 문제 3-7]

- 커널 중앙의 1을 -1로 바꾸고 커널을 중앙 평균화
- 다시 필터링하고 그래프를 작성

SECTION 03 연습 문제(3)

3.4.3 k-평균 연습 문제

[연습 문제 3-8]

- 새 데이터를 생성하지 않고 $k = 3$을 사용하여 k-평균 코드를 여러 번 실행하고 클러스터 결과가 유사한지 확인

[연습 문제 3-9]

- $k = 2$와 $k = 4$로 두고 여러 번 클러스터링을 실행해 보고 결과를 확인