CHAPTER 2
데이터 다루기

인공지능 · Lecture Note

대전대학교 컴퓨터공학과
조교수 박상돈

목차 (Table of Contents)

CHAPTER 02 데이터 다루기

이 챕터의 부제는 "수상한 생선을 조심하라!" 입니다. CHAPTER 01에서 k-최근접 이웃 알고리즘으로 도미와 빙어를 완벽하게 분류하는 데 성공했지만, 사실 그 과정에는 중요한 문제가 숨어 있었습니다. 데이터를 어떻게 준비하느냐에 따라 모델 성능이 극적으로 달라질 수 있다는 것을 이번 챕터에서 배우게 됩니다.

학습 목표

1. 머신러닝 알고리즘에 주입할 데이터를 준비하는 방법을 배웁니다.
2. 데이터의 형태가 알고리즘에 미치는 영향을 이해합니다.

SECTION 2-1 훈련 세트와 테스트 세트

2.1.1 지도 학습과 비지도 학습

지도 학습 (Supervised Learning)

지도 학습 알고리즘은 훈련하기 위한 데이터와 정답(타깃)이 필요합니다. CHAPTER 01에서 도미와 빙어를 분류할 때, 생선의 길이와 무게(입력)를 알고리즘에 제공하고, 각 생선이 도미인지 빙어인지(정답)를 함께 주었습니다. 이것이 바로 지도 학습입니다.

지도 학습의 핵심 용어

용어	영문	설명
입력	Input	알고리즘에 주입하는 데이터 (예: 생선의 길이와 무게)
타깃	Target	정답 데이터 (예: 도미=1, 빙어=0)
훈련 데이터	Training Data	입력과 타깃을 합친 데이터
특성	Feature	입력으로 사용된 개별 성질 (예: 길이, 무게)

지도 학습은 정답(타깃)이 있으므로 알고리즘이 정답을 맞히는 것을 학습합니다. CHAPTER 01에서 사용한 k-최근접 이웃 알고리즘은 입력 데이터와 타깃을 모두 사용했으므로 지도 학습 알고리즘입니다.

비지도 학습 (Unsupervised Learning)

비지도 학습 알고리즘은 타깃 없이 입력 데이터만 사용합니다. 정답을 사용하지 않으므로 무언가를 맞힐 수는 없지만, 대신 데이터를 잘 파악하거나 변형하는 데 도움이 됩니다. 비지도 학습의 대표적인 예로는 군집(Clustering)과 차원 축소가 있으며, CHAPTER 06에서 다룹니다.

구분	지도 학습	비지도 학습
타깃(정답)	필요함	없음
목적	입력에 대한 정답 예측	데이터의 패턴·구조 파악
예시	분류, 회귀	군집, 차원 축소
본 교재 범위	CH01~CH05	CH06

참고: 머신러닝은 지도 학습, 비지도 학습 외에 강화 학습(Reinforcement Learning)으로도 나뉩니다. 강화 학습은 에이전트가 환경과 상호작용하며 보상을 최대화하는 방식으로 학습하며, 본 교재에서는 다루지 않습니다.

2.1.2 훈련 세트와 테스트 세트의 필요성

왜 데이터를 나눠야 하는가?

CHAPTER 01에서 도미와 빙어를 100% 완벽하게 분류했습니다. 그런데 여기에는 치명적인 문제가 있습니다. 훈련에 사용한 데이터와 동일한 데이터로 모델을 평가했다는 것입니다.

비유: *중간고사를 보기 전에 출제될 시험 문제와 정답을 미리 알려주고 시험을 본다면? 100점을 맞는 것이 당연하겠죠. 하지만 그것이 진짜 실력일까요?*

머신러닝도 마찬가지입니다. 도미와 빙어의 데이터와 타깃을 주고 훈련한 다음, 같은 데이터로 테스트하면 모두 맞히는 것이 당연합니다. 머신러닝 알고리즘의 성능을 제대로 평가하려면 훈련 데이터와 평가에 사용할 데이터가 달라야 합니다.

훈련 세트와 테스트 세트

구분	훈련 세트 (Train Set)	테스트 세트 (Test Set)
용도	모델을 훈련시키는 데 사용	훈련된 모델의 성능을 평가
비율	전체 데이터의 70~80%	전체 데이터의 20~30%
핵심 원칙	데이터가 클수록 좋음	훈련에 사용하지 않은 데이터여야 함

가장 간단한 방법은 이미 준비된 데이터 중에서 일부를 떼어 내어 테스트 세트로 활용하는 것입니다. 이것이 가장 일반적인 평가 방법입니다.

2.1.3 샘플링 편향 (Sampling Bias)

잘못된 데이터 분리의 예

CHAPTER 01의 데이터(도미 35개 + 빙어 14개 = 총 49개)에서 처음 35개를 훈련 세트로, 나머지 14개를 테스트 세트로 단순히 나누면 어떻게 될까요?

```
train_input = fish_data[:35]    # 처음 35개 → 전부 도미!
train_target = fish_target[:35]
test_input = fish_data[35:]     # 나머지 14개 → 전부 빙어!
test_target = fish_target[35:]
```

```
kn.fit(train_input, train_target)
kn.score(test_input, test_target)  # 결과: 0.0
```

정확도가 0.0! 훈련 세트에는 도미만 들어 있고, 테스트 세트에는 빙어만 들어 있기 때문입니다. 빙어를 한 번도 본 적 없는 모델이 빙어를 맞힐 수 있을 리가 없습니다.

샘플링 편향이란

샘플링 편향(Sampling Bias)이란 훈련 세트와 테스트 세트에 샘플이 골고루 섞여 있지 않아 한쪽으로 치우친 현상을 말합니다. 훈련 세트나 테스트 세트에 특정 클래스(도미 또는 빙어)의 데이터만 들어가 있으면 올바른 학습이 이루어지지 않습니다.

> ✔ **핵심 원칙**
> 훈련 세트와 테스트 세트에는 각 클래스의 데이터가 골고루 섞여 있어야 합니다. 이를 위해 데이터를 무작위로 섞은 뒤 나누는 것이 중요합니다.

2.1.4 넘파이(NumPy)를 활용한 데이터 분리

넘파이란

넘파이(NumPy)는 파이썬의 대표적인 배열(array) 라이브러리입니다. 고차원의 배열을 손쉽게 만들고 조작할 수 있는 간편한 도구를 많이 제공합니다. 머신러닝에서 데이터를 다룰 때 반드시 사용하게 되는 핵심 라이브러리입니다.

파이썬 리스트를 넘파이 배열로 변환

```
import numpy as np

input_arr = np.array(fish_data)
target_arr = np.array(fish_target)
```

```
print(input_arr.shape)   # (49, 2) → 49개 샘플, 2개 특성
```

샘플(sample): 하나의 생선 데이터를 의미합니다. 도미 35마리 + 빙어 14마리 = 전체 49개의 샘플. 사용하는 특성은 길이와 무게 2개입니다.

인덱스를 섞어 데이터 분리하기

배열을 직접 섞는 대신, 인덱스를 섞은 다음 그 인덱스로 input_arr와 target_arr에서 샘플을 선택하는 방법을 사용합니다. 이렇게 하면 입력 데이터와 타깃 데이터가 항상 같은 위치에서 함께 선택됩니다.

```
np.random.seed(42)           # 재현 가능한 결과를 위해 시드 고정
index = np.arange(49)        # 0~48까지 인덱스 배열 생성
np.random.shuffle(index)     # 인덱스를 무작위로 섞기
```

```
# 배열 인덱싱: 여러 개의 인덱스로 한 번에 여러 원소를 선택
train_input = input_arr[index[:35]]    # 처음 35개 인덱스 → 훈련 세트
```

```
train_target = target_arr[index[:35]]
test_input = input_arr[index[35:]]     # 나머지 14개 인덱스 → 테스트 세트
test_target = target_arr[index[35:]]
```

산점도로 확인하면 훈련 세트(파란색)와 테스트 세트(주황색)에 도미와 빙어가 골고루 섞여 있음을 확인할 수 있습니다.

```
import matplotlib.pyplot as plt
plt.scatter(train_input[:,0], train_input[:,1])   # 훈련 세트
plt.scatter(test_input[:,0], test_input[:,1])     # 테스트 세트
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```

2.1.5 두 번째 머신러닝 프로그램

넘파이로 골고루 섞어 만든 훈련 세트와 테스트 세트로 k-최근접 이웃 모델을 다시 훈련합니다.

```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(train_input, train_target)      # 훈련
kn.score(test_input, test_target)       # 결과: 1.0 (100% 정확도!)
```

```
# 예측 결과와 실제 타깃 비교
print(kn.predict(test_input))
# array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
print(test_target)
# array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
```

예측 결과와 실제 타깃이 완전히 일치합니다! 테스트 세트에서 100%의 정확도를 달성했습니다.

> ✔ 중요
> fit() 메서드를 실행할 때마다 KNeighborsClassifier 객체는 이전에 학습한 모든 것을 잃어버립니다. 이전 모델을 유지하고 싶다면 새 객체를 만들어야 합니다. 또한 predict() 메서드가 반환하는 값은 단순 파이썬 리스트가 아닌 넘파이 배열입니다.

2.1.6 Section 2-1 정리 및 확인 문제

문제 해결 과정 요약

단계	내용
문제 인식	훈련에 사용한 데이터로 모델을 평가하면 '정답을 미리 알려주고 시험 보는 것'과 같음

해결 방법	데이터를 훈련 세트와 테스트 세트로 나누어 공정하게 평가
주의사항	데이터를 나눌 때 각 클래스가 골고루 섞여 있어야 함 (샘플링 편향 방지)
도구	넘파이의 shuffle() 함수로 인덱스를 무작위로 섞어 분리
결과	테스트 세트에서 100%의 정확도 달성

키워드 정리

- 지도 학습: 입력과 타깃을 전달하여 모델을 훈련한 다음 새로운 데이터를 예측하는 데 활용
- 비지도 학습: 타깃 데이터 없이 입력 데이터에서 특징을 찾는 데 활용
- 훈련 세트: 모델을 훈련할 때 사용하는 데이터. 클수록 좋으며, 테스트 세트를 제외한 모든 데이터
- 테스트 세트: 전체 데이터의 20~30%를 사용. 전체 데이터가 아주 크다면 1%만 덜어내도 충분

확인 문제

Q1. 샘플의 입력과 타깃(정답)을 알고 있을 때 사용할 수 있는 학습 방법은?

→ **정답: ① 지도 학습**

Q2. 훈련 세트와 테스트 세트가 전체 데이터를 대표하지 못하는 현상은?

→ **정답: ④ 샘플링 편향**

Q3. 사이킷런은 입력 데이터 배열을 어떻게 구성할 것으로 기대하는가?

→ **정답: ② 행: 샘플, 열: 특성**

Q4. 배열 arr에서 두 번째~다섯 번째 원소를 선택하는 올바른 슬라이싱은?

→ **정답: ③ arr[1:5] (인덱스 1, 2, 3, 4를 선택. 마지막 인덱스는 포함되지 않음)**

SECTION 2-2 데이터 전처리

이 섹션에서는 새로운 문제를 다룹니다. 길이가 25cm이고 무게가 150g인 생선이 있다면, 이것은 도미일까요 빙어일까요? 산점도를 보면 분명히 도미에 가까운데, k-최근접 이웃 모델은 이 생선을 빙어라고 예측합니다. 왜 이런 일이 벌어지는지, 어떻게 해결하는지 알아봅시다.

2.2.1 넘파이로 데이터 준비하기

column_stack()으로 2 차원 배열 만들기

넘파이의 column_stack() 함수는 전달받은 리스트를 일렬로 세운 다음 차례대로 나란히 연결합니다. 이전 챕터에서 zip()과 리스트 내포를 사용한 것보다 더 간편합니다.

```
import numpy as np

# 예시: 두 리스트를 나란히 붙이기
np.column_stack(([1,2,3], [4,5,6]))
# 결과: array([[1, 4],
#              [2, 5],
#              [3, 6]])
```

```
# 생선 데이터에 적용
fish_data = np.column_stack((fish_length, fish_weight))
print(fish_data[:5])
# [[ 25.4 242. ]
#  [ 26.3 290. ]
#  [ 26.5 340. ]
#  [ 29.  363. ]
#  [ 29.  430. ]]
```

타깃 데이터 만들기

np.ones()와 np.zeros() 함수로 각각 원하는 개수의 1과 0을 채운 배열을 만든 뒤, np.concatenate() 함수로 연결합니다.

```
fish_target = np.concatenate((np.ones(35), np.zeros(14)))
# [1. 1. 1. ... 1. 0. 0. ... 0.] (1이 35개, 0이 14개)
```

함수	역할	연결 방향
np.column_stack()	두 배열을 열(column) 방향으로 나란히 연결	좌우 (옆으로)
np.concatenate()	배열을 첫 번째 축(행) 방향으로 이어 붙임	상하 (아래로)
np.ones(n)	1로 채워진 길이 n의 배열 생성	-

np.zeros(n)	0으로 채워진 길이 n의 배열 생성	-

2.2.2 train_test_split()으로 데이터 나누기

사이킷런은 머신러닝 모델을 위한 알고리즘뿐만 아니라 다양한 유틸리티 도구도 제공합니다. 그 중 train_test_split() 함수는 전달되는 리스트나 배열을 비율에 맞게 훈련 세트와 테스트 세트로 나누어 줍니다.

```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, random_state=42)
```

fish_data와 fish_target 2개의 배열을 전달했으므로 2개씩 나뉘어 총 4개의 배열이 반환됩니다. 기본적으로 25%를 테스트 세트로 분리합니다.

```
print(train_input.shape, test_input.shape)   # (36, 2) (13, 2)
print(test_target)   # [1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

stratify 매개변수로 클래스 비율 유지

위 결과를 보면 도미:빙어 비율이 원본(2.5:1)과 테스트 세트(3.3:1)가 다릅니다. 무작위로 나눌 때 특히 데이터가 적으면 이런 샘플링 편향이 발생합니다.

stratify 매개변수에 타깃 데이터를 전달하면 클래스 비율에 맞게 데이터를 나눕니다. 훈련 데이터가 작거나 특정 클래스의 샘플 개수가 적을 때 특히 유용합니다.

```
train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, stratify=fish_target, random_state=42)

print(test_target)   # [0. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1.]
# 도미:빙어 = 9:4 = 2.25:1 → 원본 비율(2.5:1)에 가까움
```

> **✔ train_test_split() 핵심 매개변수**
> random_state: 랜덤 시드를 지정하여 재현 가능한 결과를 얻음. stratify: 클래스 비율을 유지하며 나눔 (분류 문제에서 항상 사용 권장). test_size: 테스트 세트의 비율 지정 (기본값 0.25).

2.2.3 수상한 도미 한 마리 — 스케일 문제

문제 발생

stratify를 사용하여 나눈 데이터로 k-최근접 이웃 모델을 훈련하고 평가합니다.

```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(train_input, train_target)
kn.score(test_input, test_target)     # 결과: 1.0
```

테스트 세트에서는 100% 정확도입니다. 그런데 길이 25cm, 무게 150g인 생선을 예측하면...

```
print(kn.predict([[25, 150]]))         # 결과: [0.] → 빙어!
```

산점도를 보면 이 샘플(25cm, 150g)은 분명히 오른쪽 위의 도미 데이터에 더 가까운데, 왜 빙어라고 판단했을까요?

원인 분석: kneighbors() 메서드

KNeighborsClassifier 클래스의 kneighbors() 메서드는 주어진 샘플에서 가장 가까운 이웃까지의 거리와 이웃 샘플의 인덱스를 반환합니다.

```
distances, indexes = kn.kneighbors([[25, 150]])

print(train_input[indexes])
# [[[ 25.4 242. ]     ← 도미 1개
#   [ 15.   19.9]     ← 빙어
#   [ 14.3  19.7]     ← 빙어
#   [ 13.   12.2]     ← 빙어
#   [ 12.2  12.2]]]   ← 빙어

print(train_target[indexes])  # [[1. 0. 0. 0. 0.]] → 5개 중 빙어 4개!

print(distances)
# [[ 92.00  130.48  130.74  138.32  138.39]]
```

스케일 차이가 원인

가장 가까운 도미까지 거리는 92이고, 나머지 빙어까지 거리는 130~138입니다. 그래프에서 거리 비율이 이상한 이유는 다음과 같습니다.

- x 축(길이)의 범위: 10~40 (약 30 차이)
- y 축(무게)의 범위: 0~1000 (약 1000 차이)

y축 범위가 x축보다 약 33배 넓습니다. 따라서 y축(무게)으로 조금만 멀어져도 거리가 아주 큰 값으로 계산됩니다. k-최근접 이웃은 거리 기반 알고리즘이므로, 생선의 길이(x축)는 거리 계산에 거의 영향을 미치지 못하고 무게(y축)만 고려 대상이 됩니다.

맷플롯립의 xlim() 함수로 x축 범위를 0~1000으로 맞추면 이 문제가 명확히 보입니다. 두 특성의 스케일(scale)이 다르기 때문에 발생한 문제입니다.

```
plt.xlim((0, 1000))   # x축 범위를 y축과 동일하게 설정
```

→ 데이터가 y축을 따라 길게 늘어지고, x축 방향의 차이는 거의 무시됩니다.

2.2.4 표준점수(Standard Score)로 전처리

데이터 전처리란

데이터 전처리(Data Preprocessing)란 데이터를 표현하는 기준이 다를 때, 특성값을 일정한 기준으로 맞춰 주는 작업입니다. 특히 k-최근접 이웃처럼 거리 기반 알고리즘은 샘플 간의 거리에 영향을 많이 받으므로 반드시 데이터 전처리가 필요합니다.

표준점수(Standard Score, z-점수)

표준점수는 각 특성값이 평균에서 표준편차의 몇 배만큼 떨어져 있는지를 나타냅니다. 이를 통해 실제 특성값의 크기와 상관없이 동일한 조건으로 비교할 수 있습니다.

공식: z = (값 - 평균) / 표준편차

```
# 각 특성(열)별 평균과 표준편차 계산
mean = np.mean(train_input, axis=0)
std = np.std(train_input, axis=0)
print(mean, std)  # [27.30  454.10] [9.98  323.30]

# 표준점수로 변환 (넘파이 브로드캐스팅)
train_scaled = (train_input - mean) / std
```

axis=0은 행을 따라 각 열의 통계 값을 계산하라는 의미입니다. 특성마다 스케일이 다르므로 평균과 표준편차는 각 특성별로 따로 계산해야 합니다.

넘파이 브로드캐스팅

브로드캐스팅(Broadcasting)은 크기가 다른 넘파이 배열에서 자동으로 사칙 연산을 모든 행이나 열로 확장하여 수행하는 기능입니다. train_input은 (36, 2) 크기이고, mean과 std는 (2,) 크기이지만, 넘파이가 자동으로 모든 행에 대해 뺄셈과 나눗셈을 수행합니다.

변환 후 x축과 y축의 범위가 대략 -1.5~1.5 사이로 통일됩니다. 이제 길이와 무게의 스케일이 동일해졌습니다.

✔ **핵심 주의사항**
데이터를 전처리할 때 훈련 세트를 변환한 방식 그대로 테스트 세트를 변환해야 합니다. 즉, 훈련 세트의 평균과 표준편차로 테스트 세트도 변환합니다. 테스트 세트의 평균/표준편차를 따로 계산하면 안 됩니다!

2.2.5 전처리 데이터로 모델 훈련하기

표준점수로 변환한 데이터로 k-최근접 이웃 모델을 다시 훈련합니다.

```
# 1. 모델 훈련 (표준점수로 변환된 훈련 세트 사용)
kn.fit(train_scaled, train_target)

# 2. 테스트 세트도 훈련 세트의 평균/표준편차로 변환
test_scaled = (test_input - mean) / std

# 3. 모델 평가
kn.score(test_scaled, test_target)     # 결과: 1.0

# 4. 문제의 도미 샘플로 다시 예측
new = ([25, 150] - mean) / std          # 같은 방식으로 변환
print(kn.predict([new]))                # 결과: [1.] → 도미!
```

이제 올바르게 도미로 예측합니다! kneighbors() 메서드로 확인하면, 이 샘플의 5개 이웃이 모두 도미로 나타납니다. 표준점수 변환을 통해 두 특성의 스케일이 동일해졌기 때문에, k-최근접 이웃 알고리즘이 올바른 거리를 계산할 수 있게 된 것입니다.

```
distances, indexes = kn.kneighbors([new])
plt.scatter(train_scaled[:,0], train_scaled[:,1])
plt.scatter(new[0], new[1], marker='^')
plt.scatter(train_scaled[indexes,0], train_scaled[indexes,1], marker='D')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```

→ 삼각형 샘플 주변의 5개 다이아몬드가 모두 도미 데이터입니다.

2.2.6 Section 2-2 정리 및 확인 문제

문제 해결 과정 요약

단계	내용
문제	도미에 가까운 샘플(25cm, 150g)을 빙어라고 엉뚱한 예측
원인	길이와 무게의 스케일이 달라 무게만으로 거리가 결정됨

해결	표준점수(z-점수)로 특성값을 변환하여 스케일 통일
핵심 주의	테스트 세트는 반드시 훈련 세트의 평균·표준편차로 변환

키워드 정리

- 데이터 전처리: 머신러닝 모델에 훈련 데이터를 주입하기 전에 가공하는 단계. 때때로 많은 시간이 소모됨
- 표준점수: 훈련 세트의 스케일을 바꾸는 대표적인 방법. 평균을 빼고 표준편차로 나눔
- 브로드캐스팅: 크기가 다른 넘파이 배열에서 자동으로 사칙 연산을 모든 행이나 열로 확장하여 수행하는 기능

확인 문제

Q1. 특성값을 0에서 표준편차의 몇 배수만큼 떨어져 있는지로 변환한 값을 무엇이라 부르는가?

→ **정답: ③ 표준점수**

Q2. 테스트 세트의 스케일을 조정하려면 어떤 데이터의 통계 값을 사용해야 하는가?

→ **정답: ① 훈련 세트 (훈련 세트의 평균과 표준편차로 테스트 세트를 변환해야 함)**

Q3. for 반복문 없이 넘파이 배열의 모든 원소에 산술 연산이 적용되는 기능은?

→ **정답: ③ 브로드캐스팅**

Q4. 다음 중 넘파이 배열 함수가 아닌 것은?

→ **정답: ② nils() (ones, mean, std는 모두 넘파이 함수)**

핵심 용어 및 패키지 정리

핵심 용어

용어	영문	설명
지도 학습	Supervised Learning	입력과 타깃(정답)을 사용하여 모델을 훈련하는 방식
비지도 학습	Unsupervised Learning	타깃 없이 입력 데이터에서 패턴을 찾는 방식
훈련 세트	Training Set	모델을 훈련하는 데 사용하는 데이터
테스트 세트	Test Set	모델의 성능을 평가하는 데 사용하는 데이터
샘플	Sample	하나의 데이터 포인트 (예: 생선 1마리)
샘플링 편향	Sampling Bias	훈련/테스트 세트가 전체 데이터를 대표하지 못하는 현상
데이터 전처리	Data Preprocessing	모델 훈련 전에 데이터를 가공하는 단계
표준점수	Standard Score (z-score)	z = (값 - 평균) / 표준편차
브로드캐스팅	Broadcasting	크기가 다른 배열에서 자동으로 연산을 확장하는 넘파이 기능

핵심 패키지와 함수

NumPy

함수	설명
np.array(list)	파이썬 리스트를 넘파이 배열로 변환
np.arange(n)	0에서 n-1까지 1씩 증가하는 배열 생성
np.random.seed(n)	난수 생성을 위한 초깃값 지정 (결과 재현용)
np.random.shuffle(arr)	배열을 무작위로 섞음 (다차원은 첫 번째 축만)
np.column_stack((a, b))	두 배열을 열 방향으로 나란히 연결
np.concatenate((a, b))	배열을 첫 번째 축 방향으로 이어 붙임
np.ones(n) / np.zeros(n)	1 또는 0으로 채워진 배열 생성
np.mean(arr, axis=0)	각 열(특성)의 평균 계산
np.std(arr, axis=0)	각 열(특성)의 표준편차 계산
arr.shape	배열의 크기를 (행, 열) 형태로 반환

scikit-learn

클래스/함수	설명	주요 매개변수
train_test_split()	데이터를 훈련 세트와 테스트 세트로 분리	random_state, stratify, test_size
KNeighborsClassifier()	k-최근접 이웃 분류 모델	n_neighbors(기본:5)
kneighbors(sample)	가장 가까운 이웃의 거리와 인덱스 반환	n_neighbors
fit(X, y)	모델 훈련	입력 데이터, 타깃 데이터
score(X, y)	모델 성능 평가 (0~1)	입력 데이터, 타깃 데이터
predict(X)	새 데이터의 정답 예측	입력 데이터 (2차원)

전체 워크플로 요약: 데이터 준비 → 전처리(표준점수) → train_test_split()으로 분리 → fit()으로 훈련 → score()로 평가 → predict()로 예측