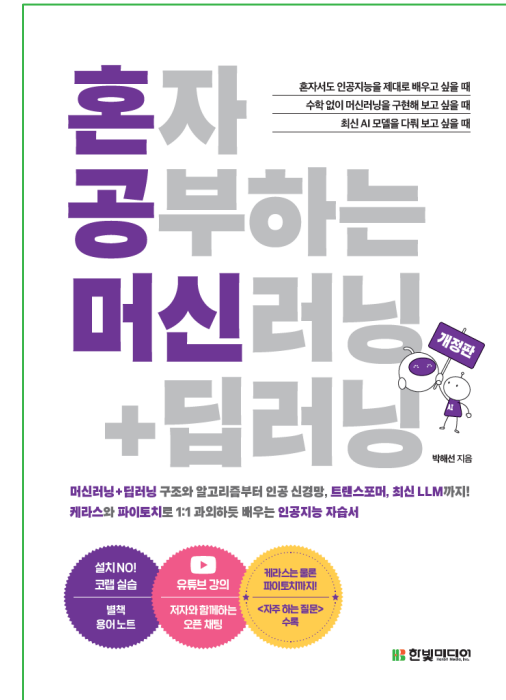


혼자 공부하는 머신러닝+딥러닝 (개정판)



한국공학대학교 게임공학과
이재영

학습 로드맵



머신러닝편

01~06장

딥러닝만 먼저 배우고
싶다면 01~04장을 읽은 후
07장으로 건너뛰어도 좋습니다.

START

01

나의 첫 머신러닝



02

데이터 다루기



03

회귀 알고리즘과 모델 규제



2번 보기

딥러닝편

07~10장

07장을 읽은 후 08장과 09장은
순서대로 읽지 않아도 괜찮습니다. 10
장을 읽기 전에 07장과 09장을
읽는 것이 좋습니다.

난이도

06

비지도 학습



05

트리 알고리즘



04

다양한 분류 알고리즘



07

딥러닝을 시작합니다



08

이미지를 위한 인공 신경망



09

텍스트를 위한 인공 신경망

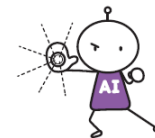


10

언어 모델을 위한 신경망



GOAL



이 책의 학습 목표

- **CHAPTER 01: 나의 첫 머신러닝**
 - 인공지능, 머신러닝, 딥러닝의 차이점을 이해합니다.
 - 구글 코랩 사용법을 배웁니다.
 - 첫 번째 머신러닝 프로그램을 만들고 머신러닝의 기본 작동 원리를 이해합니다.
- **CHAPTER 02: 데이터 다루기**
 - 머신러닝 알고리즘에 주입할 데이터를 준비하는 방법을 배웁니다.
 - 데이터 형태가 알고리즘에 미치는 영향을 이해합니다.
- **CHAPTER 03: 회귀 알고리즘과 모델 규제**
 - 지도 학습 알고리즘의 한 종류인 회귀 알고리즘에 대해 배웁니다.
 - 다양한 선형 회귀 알고리즘의 장단점을 이해합니다.
- **CHAPTER 04: 다양한 분류 알고리즘**
 - 로지스틱 회귀, 확률적 경사 하강법과 같은 분류 알고리즘을 배웁니다.
 - 이진 분류와 다중 분류의 차이를 이해하고 클래스별 확률을 예측합니다.
- **CHAPTER 05: 트리 알고리즘**
 - 성능이 좋고 이해하기 쉬운 트리 알고리즘에 대해 배웁니다.
 - 알고리즘의 성능을 최대화하기 위한 하이퍼파라미터 튜닝을 실습합니다.
 - 여러 트리를 합쳐 일반화 성능을 높일 수 있는 앙상블 모델을 배웁니다.

이 책의 학습 목표

- **CHAPTER 06: 비지도 학습**

- 타깃이 없는 데이터를 사용하는 비지도 학습과 대표적인 알고리즘을 소개합니다.
- 대표적인 군집 알고리즘인 k-평균과 DBSCAN을 배웁니다.
- 대표적인 차원 축소 알고리즘인 주성분 분석(PCA)을 배웁니다.

- **CHAPTER 07: 딥러닝을 시작합니다**

- 딥러닝의 핵심 알고리즘인 인공 신경망을 배웁니다.
- 대표적인 인공 신경망 라이브러리인 텐서플로와 케라스를 소개합니다.
- 인공 신경망 모델의 훈련을 돕는 도구를 익힙니다.

- **CHAPTER 08: 이미지를 위한 인공 신경망**

- 이미지 분류 문제에 뛰어난 성능을 발휘하는 합성곱 신경망의 개념과 구성 요소에 대해 배웁니다.
- 케라스 API로 합성곱 신경망을 만들어 패션 MNIST 데이터에서 성능을 평가해 봅니다.
- 합성곱 층의 필터와 활성화 출력을 시각화하여 합성곱 신경망이 학습한 내용을 고찰해 봅니다.

- **CHAPTER 09: 텍스트를 위한 인공 신경망**

- 텍스트와 시계열 데이터 같은 순차 데이터에 잘 맞는 순환 신경망의 개념과 구성 요소에 대해 배웁니다.
- 케라스 API로 기본적인 순환 신경망에서 고급 순환 신경망을 만들어 영화 감상평을 분류하는 작업에 적용해 봅니다.
- 순환 신경망에서 발생하는 문제점과 이를 극복하기 위한 해결책을 살펴봅니다.

CHAPTER 02 데이터 다루기

SECTION 2-1 훈련 세트와 테스트 세트

SECTION 2-2 데이터 전처리



CHAPTER 02 데이터 다루기

수상한 생선을 조심하라!

학습목표

- 머신러닝 알고리즘에 주입할 데이터를 준비하는 방법을 배웁니다.
- 데이터 형태가 알고리즘에 미치는 영향을 이해합니다.

SECTION 2-1 훈련 세트와 테스트 세트(1)

지도 학습(supervised learning)과 비지도 학습(unsupervised learning)

- 지도 학습 알고리즘은 훈련하기 위한 데이터와 정답이 필요
 - 1장 2절의 '마켓과 머신러닝'에서 보았던 도미와 빙어의 예를 보면 생선의 길이와 무게를 알고리즘에 사용
 - 이 경우 정답은 도미인지 아닌지 여부
- 지도 학습의 용어
 - 입력(input): 데이터
 - 타겟(target): 정답
 - 훈련 데이터(training data): 입력과 타겟
 - 특성(feature): 입력으로 사용된 길이와 무게 등
- 지도 학습은 정답(타겟)이 있으니 알고리즘이 정답을 맞히는 것을 학습
 - 예를 들어 도미인지 빙어인지 구분
- 비지도 학습 알고리즘은 타겟 없이 입력 데이터만 사용
 - 이런 종류의 알고리즘은 정답을 사용하지 않으므로 무언가를 맞힐 수가 없음
 - 대신 데이터를 잘 파악하거나 변형하는 데 도움이 됨
 - 비지도 학습은 6장에서 학습

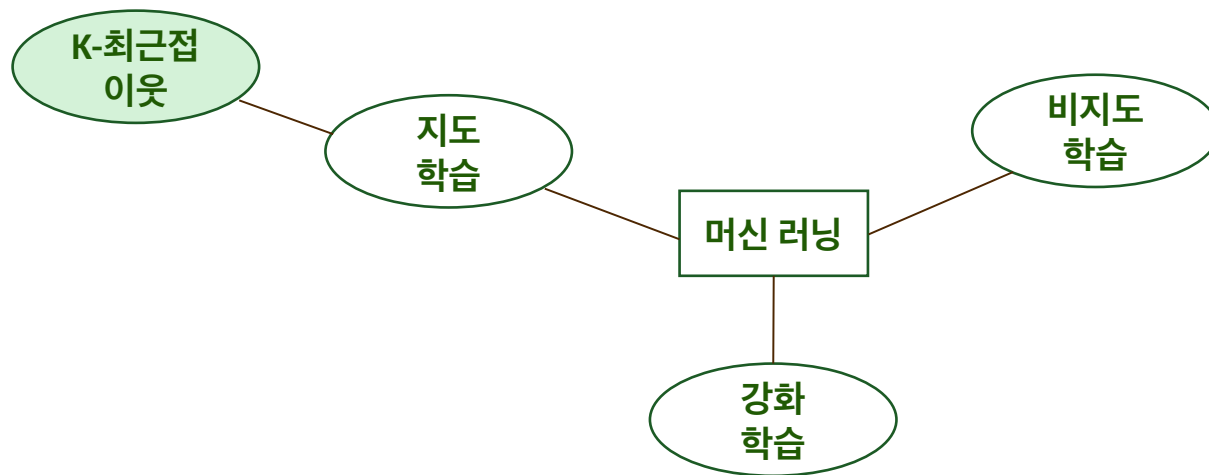
훈련 데이터

입력		타겟
길이 특성	무게 특성	
25.4	242.0	1
26.3	290.0	1
⋮	⋮	⋮
15.0	19.9	0

49개의 생선

SECTION 2-1 훈련 세트와 테스트 세트(2)

- 지도 학습(supervised learning)과 비지도 학습(unsupervised learning)
 - 1장에서 도미와 빙어를 구분하기 위해 사용한 k-최근접 이웃 알고리즘은 입력 데이터와 타깃(정답)을 사용했으므로 당연히 지도 학습 알고리즘
 - 이 알고리즘을 훈련하여 생선이 도미인지 아닌지를 판별하고, 이 모델이 훈련 데이터에서 도미를 100% 완벽하게 판별
 - 모든 것이 잘 된 것 같은데 무엇이 문제일까?



SECTION 2-1 훈련 세트와 테스트 세트(3)

◦ 훈련 세트와 테스트 세트

- 중간고사를 보기 전에 출제될 시험 문제와 정답을 미리 알려주고 시험을 본다면?
- 머신러닝도 이와 마찬가지로 도미와 빙어의 데이터와 타깃을 주고 훈련한 다음, 같은 데이터로 테스트한다면 모두 맞는 것이 당연
- 머신러닝 알고리즘의 성능을 제대로 평가하려면 훈련 데이터와 평가에 사용할 데이터가 달라야 함
- 가장 간단한 방법
 - 평가를 위해 또 다른 데이터를 준비
 - 또는, 이미 준비된 데이터 중에서 일부를 떼어 내어 활용 - 일반적 평가 방법
- 테스트 세트(test set): 평가에 사용하는 데이터
- 훈련 세트(train set): 훈련에 사용되는 데이터

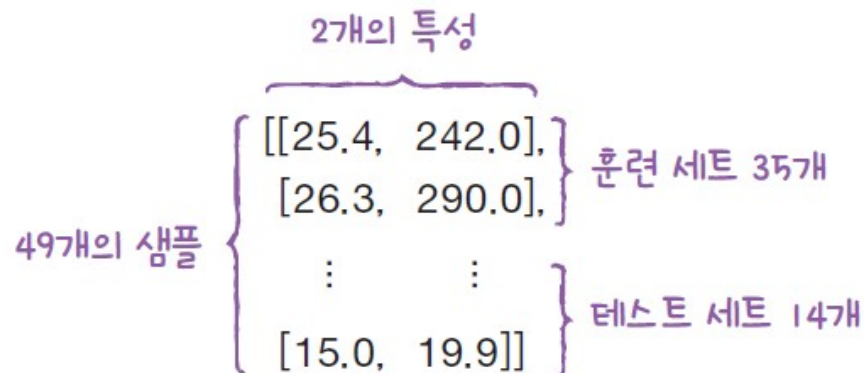
SECTION 2-1 훈련 세트와 테스트 세트(4)

◦ 훈련 세트와 테스트 세트

- 훈련할 때 사용하지 않은 데이터로 평가하기 위해, 훈련 데이터에서 일부를 떼어 내어 테스트 세트로 사용
 - 먼저 1장에서처럼 도미와 빙어의 데이터를 합쳐 하나의 파이썬 리스트로 준비
 - 1장 3절과 같이 생선의 길이와 무게를 위한 리스트를 준비
 - 손코딩 소스 http://bit.ly/bream_smelt에서 복사하여 학습에 활용
 - 두 파이썬 리스트를 순회하면서 각 생선의 길이와 무게를 하나의 리스트로 담은 2차원 리스트 생성

```
fish_data = [[l, w] for l, w in zip(fish_length, fish_weight)]  
fish_target = [1]*35 + [0]*14
```

- 샘플(sample): 하나의 생선 데이터
- 도미와 빙어는 각각 35마리, 14마리가 있으므로
전체 데이터는 49개의 샘플
- 사용하는 특성은 길이와 무게 2개
- 이 데이터의 처음 35개를 훈련 세트로, 나머지 14개를
테스트 세트로 사용



SECTION 2-1 훈련 세트와 테스트 세트(5)

◦ 훈련 세트와 테스트 세트

- 훈련할 때 사용하지 않은 데이터로 평가하기 위해, 훈련 데이터에서 일부를 떼어 내어 테스트 세트로 사용

- 먼저 사이킷런의 `KNeighborsClassifier` 클래스를 임포트하고 모델 객체를 생성

```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
```

- 전체 데이터에서 처음 35개를 선택 - 슬라이싱 연산으로 인덱스의 범위를 지정하여 원소를 여러 개 선택

- `print(fish_data[0:5])` → `[[25.4, 242.0], [26.3, 290.0], [26.5, 340.0], [29.0, 363.0], [29.0, 430.0]]`
슬라이싱 연산에서는 마지막 인덱스의 원소는 포함되지 않는다는 점을 주의

- 생선 데이터에서 처음 35개와 나머지 14개를 선택

```
# 훈련 세트로 입력값 중 0부터 34번째 인덱스까지 사용
train_input = fish_data[:35]
# 훈련 세트로 타깃값 중 0부터 34번째 인덱스까지 사용
train_target = fish_target[:35]
# 테스트 세트로 입력값 중 35번째부터 마지막 인덱스까지 사용
test_input = fish_data[35:]
# 테스트 세트로 타깃값 중 35번째부터 마지막 인덱스까지 사용
test_target = fish_target[35:]
```

SECTION 2-1 훈련 세트와 테스트 세트(6)

○ 훈련 세트와 테스트 세트

- 훈련할 때 사용하지 않은 데이터로 평가하기 위해, 훈련 데이터에서 일부를 떼어 내어 테스트 세트로 사용
 - 훈련 세트로 `fit()` 메서드를 호출해 모델을 훈련하고, 테스트 세트로 `score()` 메서드를 호출해 평가

```
kn = kn.fit(train_input, train_target)  
kn.score(test_input, test_target)
```

→ 0.0

- 정확도가 0.0?
- 혼공머신이 무엇을 잘못된 것일까?



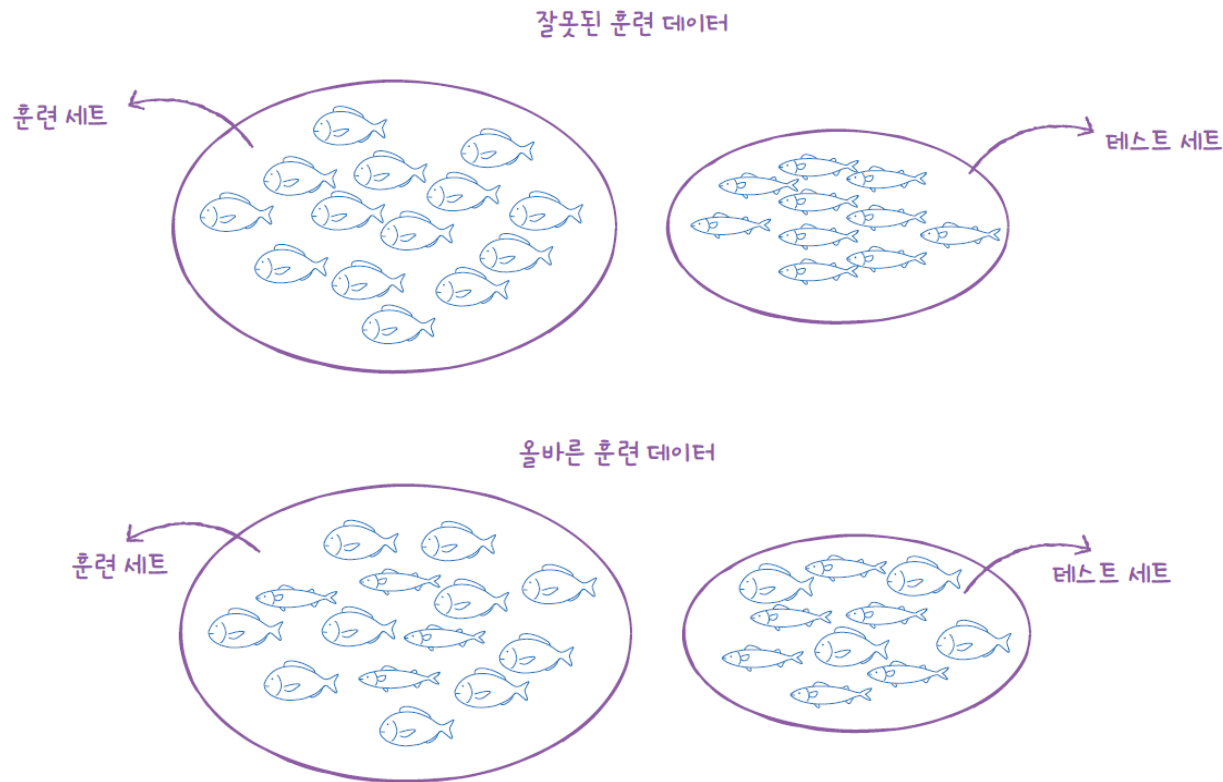
코랩에서 코드 셀을 만들고 바로 실행하는 방법

- 코드 셀에서 입력을 끝낸 다음 바로 `Alt + Enter` 키를 누르면 바로 실행하고 그 아래 새 코드 셀을 만들어 줌

SECTION 2-1 훈련 세트와 테스트 세트(7)

◦ 샘플링 편향(sampling bias)

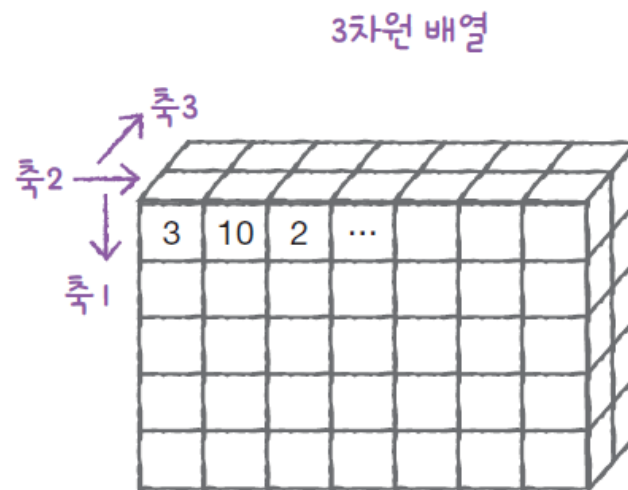
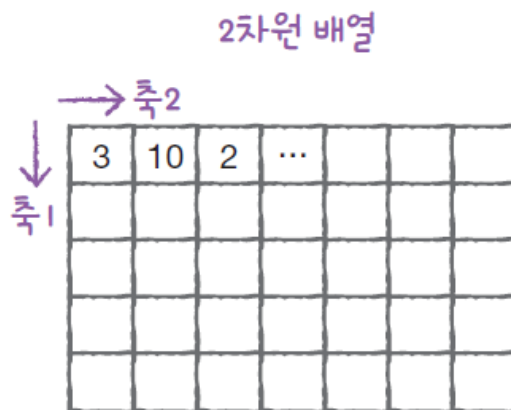
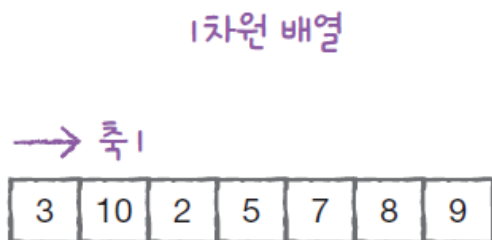
- 훈련 세트와 테스트 세트에 샘플이 골고루 섞여 있지 않아 샘플링이 한쪽으로 치우침
- 마지막 14개를 테스트 세트로 떼어 놓으면 훈련 세트에는 빙어가 하나도 들어 있지 않아, 빙어 없이 모델을 훈련하면 빙어를 올바르게 분류할 수가 없음



SECTION 2-1 훈련 세트와 테스트 세트(8)

◦ 넘파이(numpy)

- 파이썬의 대표적인 배열(array) 라이브러리
- 넘파이는 고차원의 배열을 손쉽게 만들고 조작할 수 있는 간편한 도구를 많이 제공
- 보통의 xy 좌표계와는 달리 시작점이 왼쪽 아래가 아니고 왼쪽 위에서부터 시작



SECTION 2-1 훈련 세트와 테스트 세트(9)

◦ 넘파이(numpy)

- 넘파이 라이브러리 импорт

```
import numpy as np
```

- 넘파이 `array()` 함수에 파이썬 리스트를 전달

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

- `input_arr` 배열을 출력

```
print(input_arr)
```

→

```
[[ 25.4 242. ]
 [ 26.3 290. ]
 ...
 [ 15. 19.9]]
```

- 넘파이는 배열의 차원을 구분하기 쉽도록 행과 열을 가지런히 출력
출력 결과에서 49개의 행과 2개의 열을 쉽게 확인
- 배열의 크기를 알려주는 `shape` 속성

```
print(input_arr.shape)
# 이 명령을 사용하면 (샘플 수, 특성 수)를 출력
```

→ (49, 2)

2개의 열(특성)

49개의 행 (샘플)

```
[[25.4, 242.0],
 [26.3, 290.0],
  ⋮      ⋮
 [15.0, 19.9]]
```

SECTION 2-1 훈련 세트와 테스트 세트(10)

◦ 넘파이(numpy)

- 이 배열에서 랜덤하게 샘플을 선택해 훈련 세트와 테스트 세트로 만들기
 - 배열을 섞은 후에 나누는 방식 대신에 무작위로 샘플을 고르는 방법을 사용
 - `input_arr`와 `target_arr`에서 같은 위치는 함께 선택되어야 함에 주의
- 인덱스를 섞은 다음 `input_arr`와 `target_arr`에서 샘플을 선택하면 무작위로 훈련 세트를 나누게 됨
 - 넘파이 `arange()` 함수를 사용하면 0에서부터 48까지 1씩 증가하는 인덱스를 간단히 만들 수 있음
 - 다음으로 이 인덱스를 랜덤하게 섞기

```
np.random.seed(42)
index = np.arange(49)
np.random.shuffle(index)
```

- 넘파이 `arange()` 함수에 정수 `N`을 전달하면 0에서부터 `N-1`까지 1씩 증가하는 배열을 생성
- 넘파이 `random` 패키지 아래에 있는 `shuffle()` 함수는 주어진 배열을 무작위로 섞음
- 만들어진 인덱스를 출력

```
print(index)
```



```
[13 45 47 44 17 27 26 25 31 19 12 4 34 8 3 6 40 41 46 15 9 16 24 33
 30 0 43 32 5 29 11 36 1 21 2 37 35 23 39 10 22 18 48 20 7 42 14 28
 38]
```


SECTION 2-1 훈련 세트와 테스트 세트(11)

- 넘파이(numpy)

- 랜덤하게 섞인 인덱스를 사용해 전체 데이터를 훈련 세트와 테스트 세트로 나누기
- 배열 인덱싱(array indexing): 1개의 인덱스가 아닌 여러 개의 인덱스로 한 번에 여러 개의 원소를 선택

```
print(input_arr[[1,3]])
```

→

```
[[ 26.3 290. ]  
 [ 29. 363. ]
```

- 리스트 대신 넘파이 배열을 인덱스로 전달
- 앞의 index 배열 처음 35개를 input_arr와 target_arr에 전달하여 랜덤하게 35개의 샘플을 훈련 세트로 생성

```
train_input = input_arr[index[:35]]  
train_target = target_arr[index[:35]]
```

- 나머지 14개를 테스트 세트로 생성

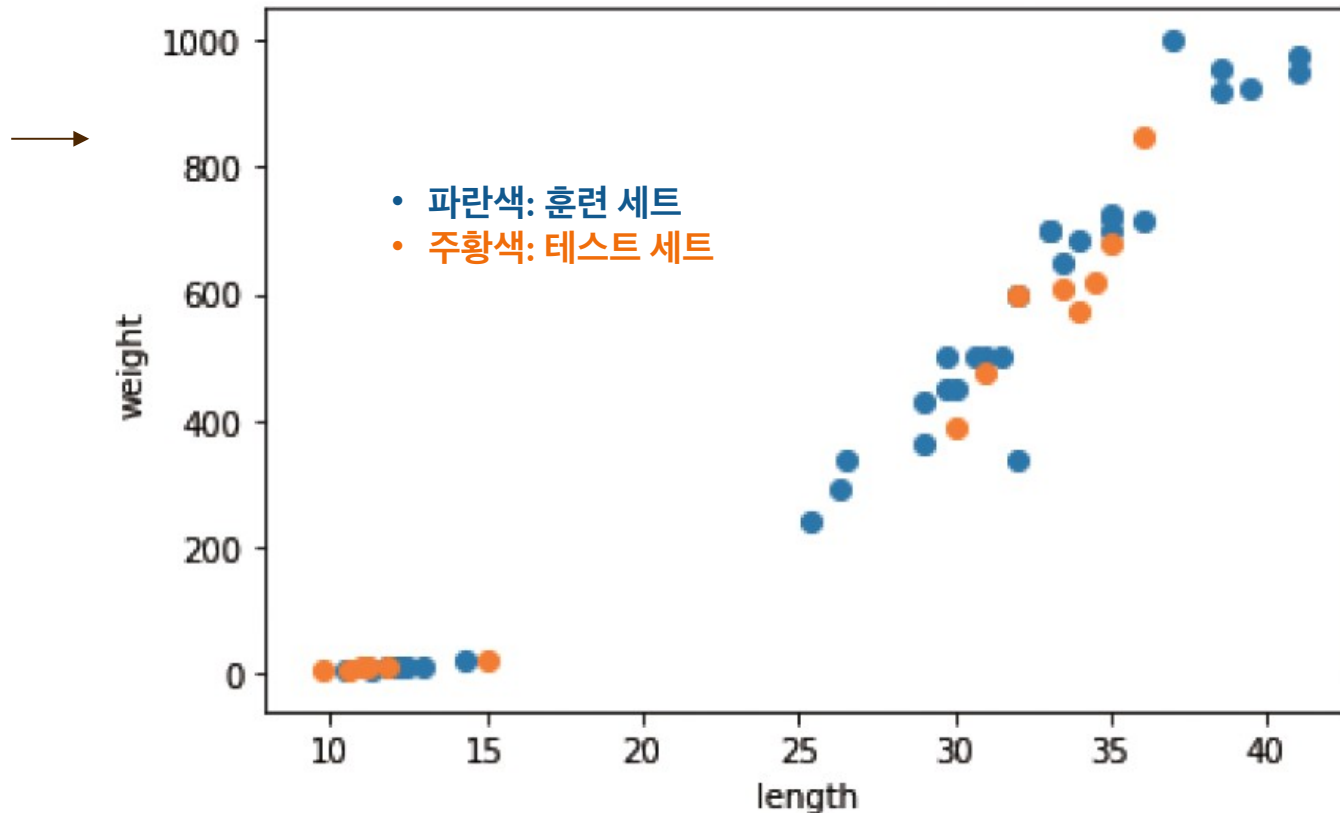
```
test_input = input_arr[index[35:]]  
test_target = target_arr[index[35:]]
```

SECTION 2-1 훈련 세트와 테스트 세트(12)

- 넘파이(numpy)

- 훈련 세트와 테스트 세트에 도미와 빙어가 잘 섞여 있는지 산점도로 확인

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



SECTION 2-1 훈련 세트와 테스트 세트(13)

- 두 번째 머신러닝 프로그램

- 앞서 만든 훈련 세트와 테스트 세트로 k-최근접 이웃 모델을 훈련
- `fit()` 메서드를 실행할 때마다 `KNeighborsClassifier` 클래스의 객체는 이전 학습한 모든 것을 잃어버림
- 이전 모델을 그대로 두고 싶다면 `KNeighborsClassifier` 클래스 객체를 새로 만들어야 함
- 여기에서는 단순히 이전에 만든 `kn` 객체를 그대로 사용

- 인덱스를 섞어 만든 `train_input`과 `train_target`으로 모델을 훈련

```
kn = kn.fit(train_input, train_target)
```

- `test_input`과 `test_target`으로 이 모델을 테스트

```
kn.score(test_input, test_target) → 1.0
```

- `predict()` 메서드로 테스트 세트의 예측 결과와 실제 타깃을 확인

```
kn.predict(test_input) → array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
```

```
test_target → array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
```

- `predict()` 메서드가 반환하는 값은 단순한 파이썬 리스트가 아닌 넘파이 배열

SECTION 2-1 훈련 세트와 테스트 세트(14)

◦ 훈련 모델 평가(문제해결 과정)

- 문제

- 알고리즘이 도미와 빙어를 모두 외우고 있다면 같은 데이터로 모델을 평가하는 것은 이상하지 않은가?
- 모델을 훈련할 때 사용한 데이터로 모델의 성능을 평가하는 것은 정답을 미리 알려주고 시험을 보는 것과 같음

- 해결

- 공정하게 점수를 매기기 위해서는 훈련에 참여하지 않은 샘플을 사용
- 훈련 데이터를 훈련 세트와 테스트 세트로 나누어, 훈련 세트로는 모델을 훈련하고 테스트 세트로 모델을 평가
- 훈련 세트나 테스트 세트에 어느 한 생선만 들어가 있다면 올바른 학습이 이루어지지 않음
- 도미와 빙어를 골고루 섞어 나누기 위해 파이썬의 다차원 배열 라이브러리인 넘파이를 사용
- 넘파이는 파이썬의 리스트와 비슷하지만 고차원의 큰 배열을 효과적으로 다룰 수 있고 다양한 도구를 많이 제공
- 이 절에서는 넘파이의 `shuffle()` 함수를 사용해 배열의 인덱스를 섞었음

- 결과

- 테스트 세트에서 100%의 정확도를 달성

SECTION 2-1 마무리(1)

- 키워드로 끝내는 핵심 포인트

- 지도 학습은 입력과 타겟을 전달하여 모델을 훈련한 다음 새로운 데이터를 예측하는 데 활용
 - 1장에서부터 사용한 k-최근접 이웃이 지도 학습 알고리즘
- 비지도 학습은 타겟 데이터가 없음
 - 무엇을 예측하는 것이 아니라 입력 데이터에서 어떤 특징을 찾는 데 주로 활용
- 훈련 세트는 모델을 훈련할 때 사용하는 데이터
 - 보통 훈련 세트가 클수록 좋음
 - 테스트 세트를 제외한 모든 데이터를 사용
- 테스트 세트는 전체 데이터에서 20~30%를 테스트 세트로 사용하는 경우가 많음
 - 전체 데이터가 아주 크다면 1%만 덜어내도 충분

SECTION 2-1 마무리(2)

◦ 핵심 패키지와 함수

- numpy

- `seed()`는 넘파이에서 난수를 생성하기 위한 정수 초깃값을 지정
- 초깃값이 같으면 동일한 난수를 뽑을 수 있으므로 랜덤 함수의 결과를 동일하게 재현하고 싶을 때 사용
- `arange()`는 일정한 간격의 정수 또는 실수 배열을 만들고, 본 간격은 1. 매개변수가 하나이면 종료 숫자를 의미 0에서 종료 숫자까지 배열을 만듦. 종료 숫자는 배열에 포함되지 않음

```
print(np.arange(3))
```

 → [0, 1, 2]

매개변수가 2개면 시작 숫자, 종료 숫자를 의미

```
print(np.arange(1, 3))
```

 → [1, 2]

매개변수가 3개면 마지막 매개변수가 간격을 나타냄

```
print(np.arange(1, 3, 0.2))
```

 → [1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8]

- `shuffle()`은 주어진 배열을 랜덤하게 섞음 - 다차원 배열일 경우 첫 번째 축(행)에 대해서만 섞음

```
arr = np.array([[1, 2], [3, 4], [5, 6]])  
np.random.shuffle(arr)  
print(arr)
```

 →

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

SECTION 2-1 확인 문제

- 1 머신러닝 알고리즘의 한 종류로서 샘플의 입력과 타깃(정답)을 알고 있을 때 사용할 수 있는 학습 방법은 무엇인가?
① 지도 학습 ② 비지도 학습
③ 차원 축소 ④ 강화 학습
- 2 훈련 세트와 테스트 세트가 잘못 만들어져 전체 데이터를 대표하지 못하는 현상을 무엇이라고 부르나?
① 샘플링 오류 ② 샘플링 실수
③ 샘플링 편차 ④ 샘플링 편향

SECTION 2-1 확인 문제

3 사이킷런은 입력 데이터(배열)가 어떻게 구성되어 있을 것으로 기대하나?

- ① 행 : 특성, 열 : 샘플 ② 행 : 샘플, 열 : 특성
- ③ 행 : 특성, 열 : 타깃 ④ 행 : 타깃, 열 : 특성


4. 다음 중 배열 `arr`에서 두 번째 원소에서부터 다섯 번째 원소까지 선택하기 위해 올바르게 슬라이싱 연산자를 사용한 것은 무엇인가요?

- ① `arr[2:5]` ② `arr[2:6]`
- ③ `arr[1:5]` ④ `arr[1:6]`

SECTION 2-2 데이터 전처리(1)

- 새로운 문제: 길이가 25cm이고 무게가 150g이면 도미? 빙어?
- 넘파이로 데이터 준비하기
 - 생선 데이터 준비: 소스 http://bit.ly/bream_smelt에서 복사
 - 넘파이를 импорт
 - 넘파이의 `column_stack()` 함수는 전달받은 리스트를 일렬로 세운 다음 차례대로 나란히 연결 2개의 리스트를 나란히 붙이기. 연결할 리스트는 파이썬 튜플(tuple)로 전달

```
np.column_stack(([1,2,3], [4,5,6]))
```




```
array([[1, 4],  
       [2, 5],  
       [3, 6]])
```

- 동일한 방법으로 `fish_length`와 `fish_weight`를 병합

```
fish_data = np.column_stack((fish_length, fish_weight))
```

- 두 리스트가 잘 연결되었는지 처음 5개의 데이터를 확인

```
print(fish_data[:5])
```



```
[[ 25.4 242. ]  
 [ 26.3 290. ]  
 [ 26.5 340. ]  
 [ 29. 363. ]  
 [ 29. 430. ]]
```

SECTION 2-2 데이터 전처리(2)

◦ 넘파이로 데이터 준비하기

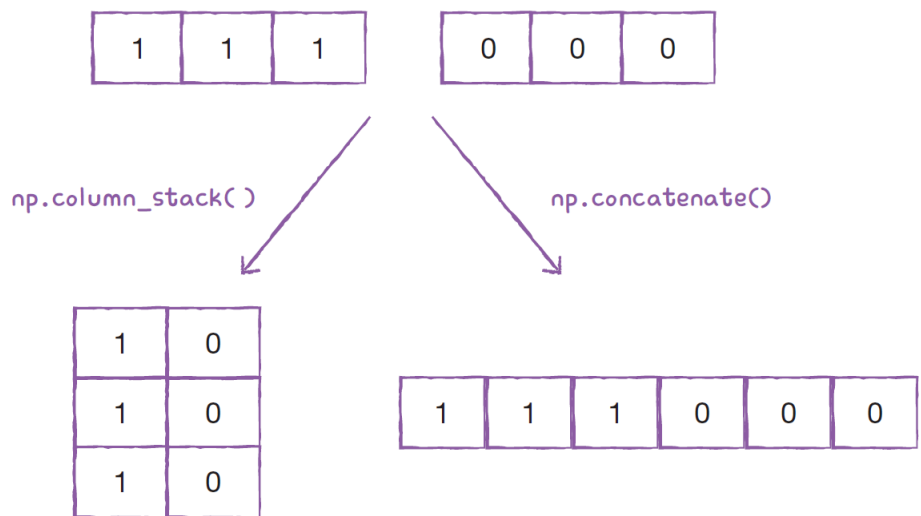
- np.ones()와 np.zeros() 함수로 타깃 데이터 만들기

- 이 두 함수는 각각 원하는 개수의 1과 0을 채운 배열을 만들어 줌

`print(np.ones(5))` → `[1. 1. 1. 1. 1.]`

- 첫 번째 차원을 따라 배열을 연결하는 np.concatenate() 함수를 사용

- 1이 35개인 배열과 0이 14개인 배열 만들기



▲ `np.column_stack()` 함수와 `np.concatenate()` 함수의 연결 방식

SECTION 2-2 데이터 전처리(3)

◉ 넘파이로 데이터 준비하기

- np.concatenate() 함수를 사용해 타깃 데이터를 만들기
np.column_stack()과 마찬가지로 연결한 리스트나 배열을 튜플로 전달해야 함

```
fish_target = np.concatenate((np.ones(35),
                               np.zeros(14)))
```

- 데이터가 잘 만들었는지 확인

[illegible]

- 데이터가 클수록 파이썬 리스트는 비효율적이므로 넘파이 배열을 사용하는 게 좋음

SECTION 2-2 데이터 전처리(4)

◦ 사이킷런으로 훈련 세트와 테스트 세트 나누기

- 사이킷런은 머신러닝 모델을 위한 알고리즘뿐만 아니라 다양한 유틸리티 도구도 제공
- `train_test_split()` 함수는 전달되는 리스트나 배열을 비율에 맞게 훈련 세트와 테스트 세트로 나누어 줌
 - `train_test_split()` 함수는 사이킷런의 `model_selection` 모듈 아래 있으며 다음과 같이 импорт

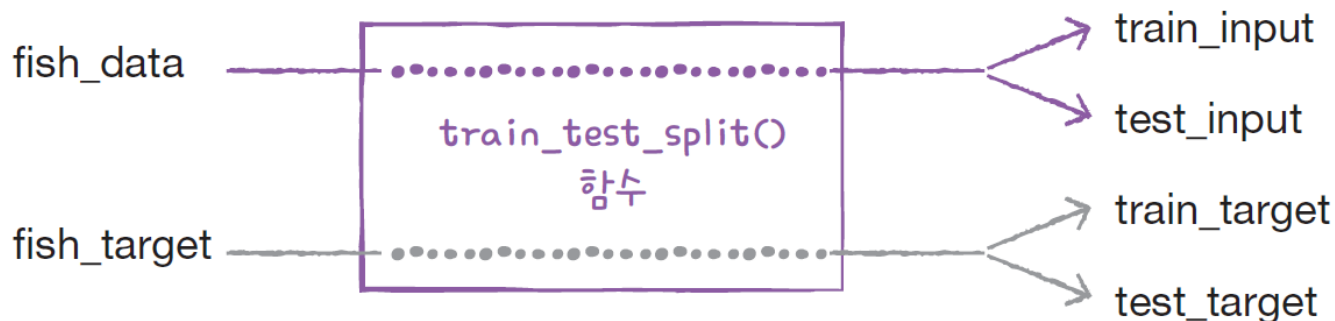
```
from sklearn.model_selection import train_test_split
```

- `fish_data`와 `fish_target` 나누기

`train_test_split()` 함수에는 자체적으로 랜덤 시드를 지정할 수 있는 `random_state` 매개변수가 있음

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

- `fish_data`와 `fish_target` 2개의 배열을 전달했으므로 2개씩 나뉘어 총 4개의 배열이 반환



SECTION 2-2 데이터 전처리(5)

- 사이킷런으로 훈련 세트와 테스트 세트 나누기

- 넘파이 배열의 `shape` 속성으로 입력 데이터의 크기를 출력

```
print(train_input.shape, test_input.shape) → (36, 2) (13, 2)
```

```
print(train_target.shape, test_target.shape) → (36,) (13,)
```

- 도미와 빙어가 잘 섞였는지 테스트 데이터를 출력

```
print(test_target) → [1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

- 도미:빙어=35 : 14=2.5 : 1 → 테스트 데이터의 도미:빙어=10 : 3=3.3 : 1 (샘플링 편향이 다소 나타남)
- 무작위로 데이터를 나누었을 때 샘플이 골고루 섞이지 않거나, 특히 일부 클래스의 개수가 적을 때 이런 일이 발생. 훈련 세트와 테스트 세트에 샘플의 클래스 비율이 일정하지 않다면 모델이 일부 샘플을 올바르게 학습할 수 없음
- `train_test_split()` 함수는 이런 문제를 간단히 해결
 `stratify` 매개변수에 타깃 데이터를 전달하면 클래스 비율에 맞게 데이터를 나눔
 훈련 데이터가 작거나 특정 클래스의 샘플 개수가 적을 때 특히 유용

SECTION 2-2 데이터 전처리(6)

- 사이킷런으로 훈련 세트와 테스트 세트 나누기

- `train_test_split()` 함수로 데이터 나누기

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

- `test_target` 출력하면 테스트 세트 비율이 2.25 : 1

```
print(test_target)    →    [0. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1.]
```

SECTION 2-2 데이터 전처리(7)

◦ 수상한 도미 한 마리

- 앞에서 준비한 데이터로 k-최근접 이웃을 훈련(1장에서 했던 것과 동일)
- 훈련 데이터로 모델을 훈련하고 테스트 데이터로 모델을 평가

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

→ 1.0

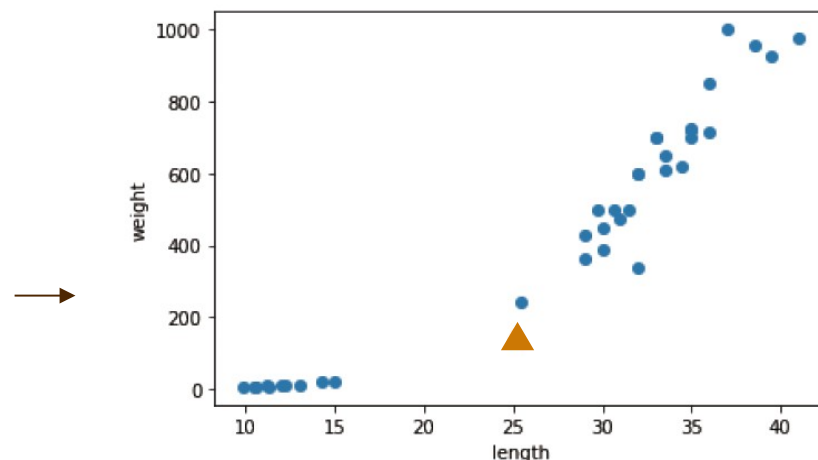
- 김 팀장이 알려준 도미 데이터를 넣고 결과를 확인(25cm, 150g)

```
print(kn.predict([[25, 150]]))
```

→ [0.]

- 이 샘플을 다른 데이터와 함께 산점도로 표시

```
import matplotlib.pyplot as plt  
plt.scatter(train_input[:,0], train_input[:,1])  
plt.scatter(25, 150, marker='^') # marker 매개변수는 모양을 지정합니다  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```



- 이 샘플은 분명히 오른쪽 위로 뻗어 있는 다른 도미 데이터에 더 가까운데, 왜 이 모델은 왼쪽 아래에 낮게 깔린 빙어 데이터에 가깝다고 판단한 걸까?

SECTION 2-2 데이터 전처리(7)

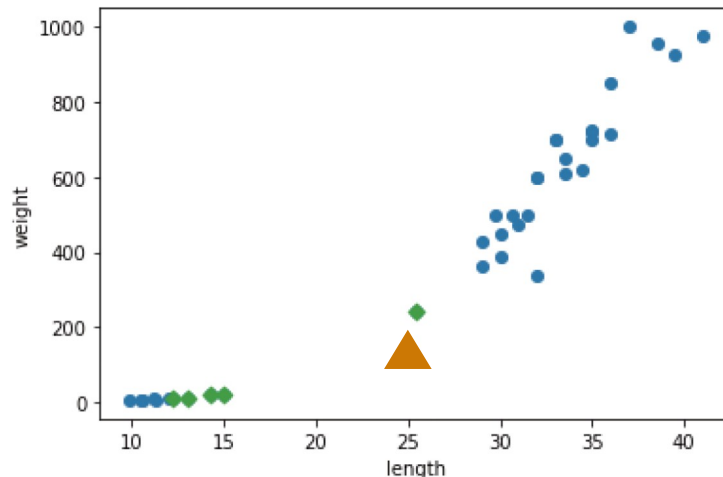
◦ 수상한 도미 한 마리

- k-최근접 이웃은 주변의 샘플 중에서 다수인 클래스를 예측으로 사용
- KNeighborsClassifier 클래스는 주어진 샘플에서 가장 가까운 이웃을 찾아 주는 kneighbors() 메서드를 제공
 - 이 메서드는 이웃까지의 거리와 이웃 샘플의 인덱스를 반환
- KNeighborsClassifier 클래스의 이웃 개수인 n_neighbors의 기본값은 5이므로 5개의 이웃이 반환됨

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

- indexes 배열을 사용해 훈련 데이터 중에서 이웃 샘플을 따로 구분해 산점도 그리기

```
plt.scatter(train_input[:,0], train_input[:,1])  
plt.scatter(25, 150, marker='^')  
plt.scatter(train_input[indexes,0],  
            train_input[indexes,1], marker='D')  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```



- ▲ 삼각형 샘플에 가장 가까운 5개의 샘플이 초록 다이아몬드로 표시

SECTION 2-2 데이터 전처리(8)

- 수상한 도미 한 마리

- 삼각형 샘플에 이웃한 데이터 확인

```
print(train_input[indexes])
```



```
[[[ 25.4 242. ]  
 [ 15. 19.9]  
 [ 14.3 19.7]  
 [ 13. 12.2]  
 [ 12.2 12.2]]]
```

- 타깃 데이터로 확인

```
print(train_target[indexes])
```



```
[[1. 0. 0. 0. 0.]]
```

- kneighbors() 메서드에서 반환한 distances 배열(이웃 샘플까지의 거리)을 출력

```
print(distances)
```

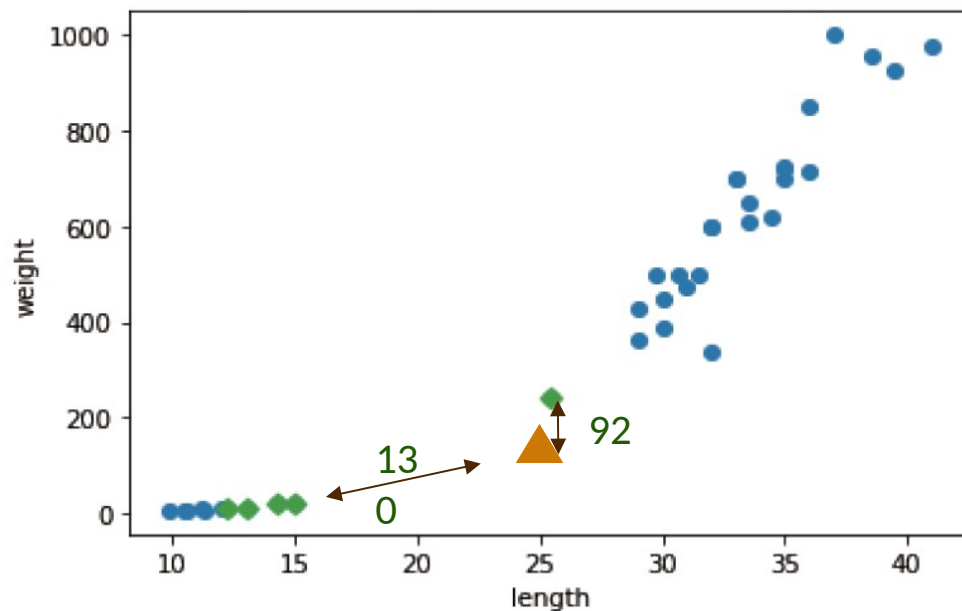


```
[[ 92.00086956  
 130.48375378  
 130.73859415  
 138.32150953  
 138.39320793]]
```

SECTION 2-2 데이터 전처리(9)

◦ 기준을 맞춰라

- 산점도에서 삼각형 샘플에 가장 가까운 첫 번째 샘플까지의 거리는 92이고, 그외 가장 가까운 샘플들은
- 모두 130, 138, 그런데 거리가 92와 130이라고 했을 때 그래프에 나타난 거리 비율이 이상하지 않은가?
- x축은 범위가 좁고(10~40), y축은 범위가 넓음(0~1000)
- 따라서 y축으로 조금만 멀어져도 거리가 아주 큰 값으로 계산
- 이 때문에 오른쪽 위의 도미 샘플이 이웃으로 선택되지 못함



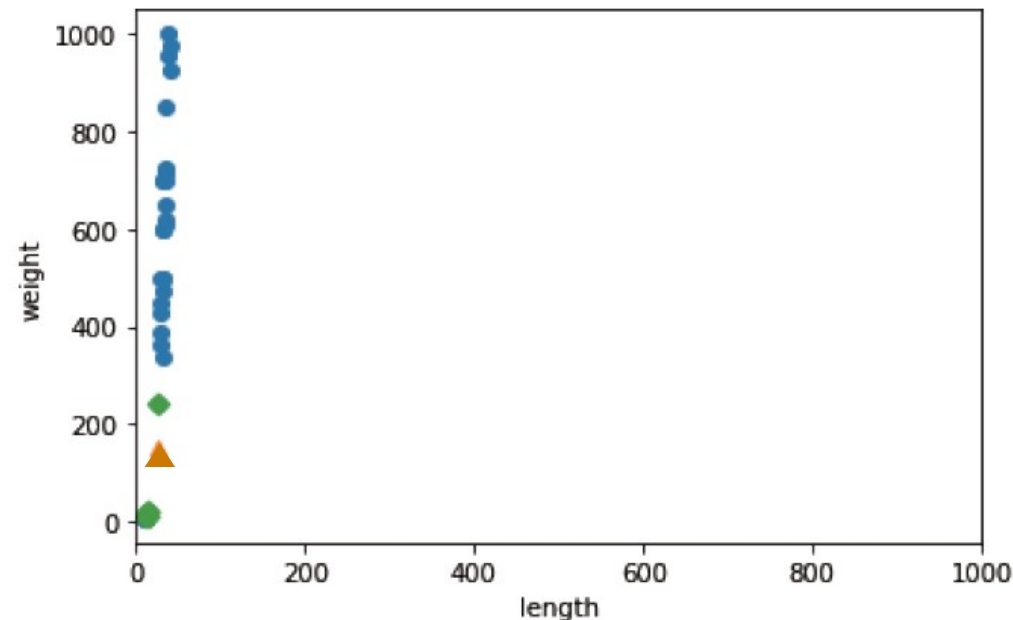
SECTION 2-2 데이터 전처리(10)

- 기준을 맞춰라

- 맷플롯립에서 `xlim()` 함수를 사용하여 x축의 범위를 동일하게 0~1,000으로 지정
(비슷하게 y축 범위를 지정하려면 `ylim()` 함수를 사용)

```
plt.scatter(train_input[:,0], train_input[:,1])  
plt.scatter(25, 150, marker='^')  
plt.scatter(train_input[indexes,0], train_input[indexes,1],  
            marker='D')  
plt.xlim((0, 1000))  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```

- 생선의 길이(x축)는 가장 가까운 이웃을 찾는 데 크게 영향을 미치지 못하고, 생선의 무게(y축)만 고려 대상이 됨
- 두 특성의 스케일(scale)이 다르기 때문



SECTION 2-2 데이터 전처리(11)

- 기준을 맞춰라

- 데이터 전처리(data preprocessing)

- 데이터를 표현하는 기준이 다르면 알고리즘이 올바르게 예측할 수 없음
알고리즘이 거리 기반일 때 특히 그러하고, k-최근접 이웃도 포함됨
 - 이런 알고리즘들은 샘플 간의 거리에 영향을 많이 받으므로 제대로 사용하기 위해 특성값을 일정한 기준으로 맞춰 주는 작업이 데이터 전처리
 - 표준점수(standard score, 혹은 z 점수): 표준점수는 각 특성값이 평균에서 표준편차의 몇 배만큼 떨어져 있는지를 나타냄. 이를 통해 실제 특성값의 크기와 상관없이 동일한 조건으로 비교

$$z = \frac{x - \mu}{\sigma}$$

- 넘파이로 평균과 표준편차 구하기

```
mean = np.mean(train_input, axis=0)
std = np.std(train_input, axis=0)
```

- np.mean() 함수는 평균을 계산하고, np.std() 함수는 표준편차를 계산. train_input은 (36, 2) 크기의 배열
 - 특성마다 값의 스케일이 다르므로 평균과 표준편차는 각 특성별로 계산. 이를 위해 axis=0으로 지정하여 행을 따라 각 열의 통계 값을 계산

SECTION 2-2 데이터 전처리(12)

◦ 기준을 맞춰라

- 데이터 전처리(data preprocessing)

- 계산된 평균과 표준편차를 출력

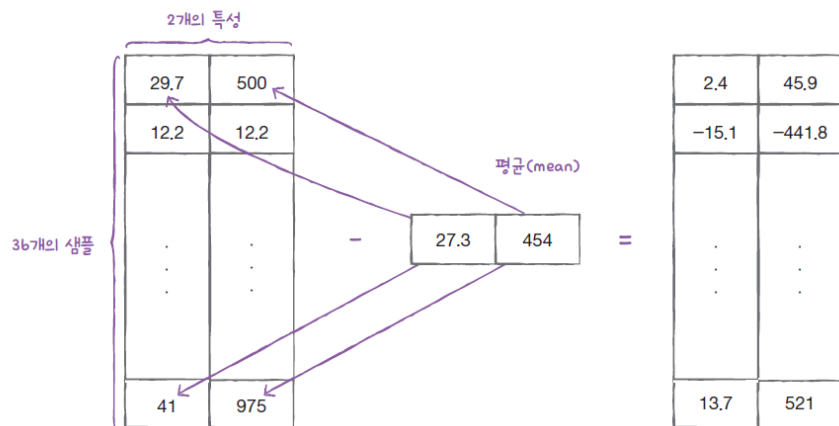
```
print(mean, std)
```

→ [27.29722222 454.09722222] [9.98244253 323.29893931]

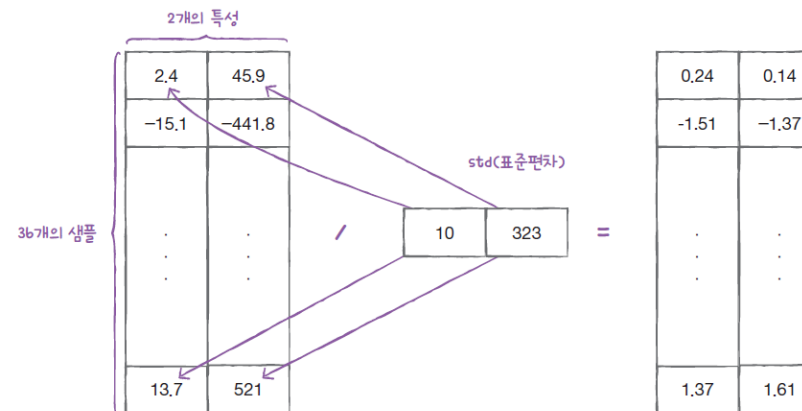
- 원본 데이터에서 평균을 빼고 표준편차로 나누어 표준점수로 변환
넘파이의 브로드캐스팅(broadcasting)

```
train_scaled = (train_input - mean) / std
```

mean(평균) 빼기



std(표준편차) 나누기

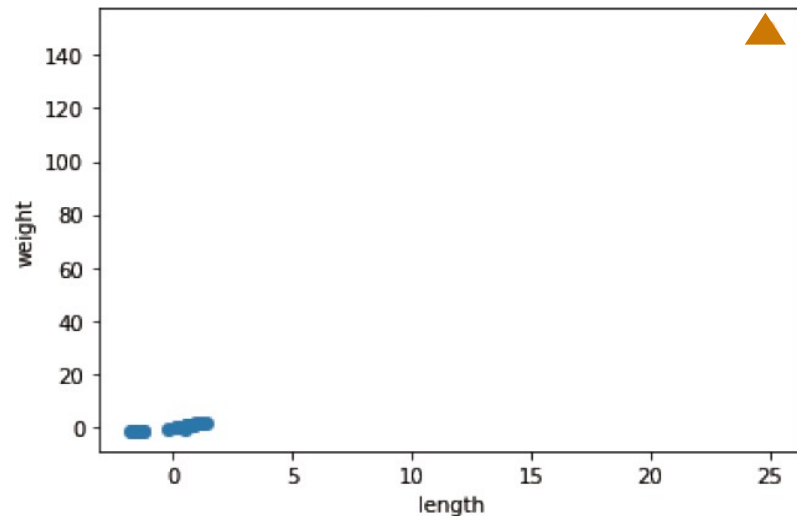


SECTION 2-2 데이터 전처리(13)

- 전처리 데이터로 모델 훈련하기

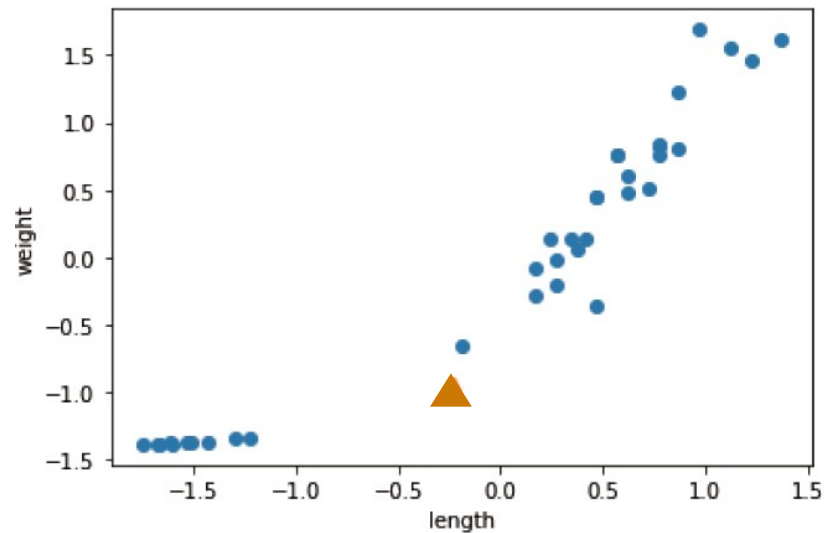
- 변환한 표준점수와 샘플의 산점도

```
plt.scatter(train_scaled[:,0], train_scaled[:,1])  
plt.scatter(25, 150, marker='^')  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```



- 샘플 [20, 150]을 동일한 비율로 변환한 산점도

```
new = ([25, 150] - mean) / std  
plt.scatter(train_scaled[:,0], train_scaled[:,1])  
plt.scatter(new[0], new[1], marker='^')  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```



x축과 y축의 범위가 -1.5~1.5 사이로 바뀜

SECTION 2-2 데이터 전처리(14)

◦ 전처리 데이터로 모델 훈련하기

- 변환한 데이터셋으로 k-최근접 이웃 모델을 다시 훈련

```
kn.fit(train_scaled, train_target)
```

- 테스트 세트도 훈련 세트의 평균과 표준편차로 변환

```
test_scaled = (test_input - mean) / std
```

- 모델 평가

```
kn.score(test_scaled, test_target)
```

→ 1.0

- 훈련 세트의 평균과 표준편차로 변환한 김 팀장의 샘플을 사용해 모델의 예측을 출력

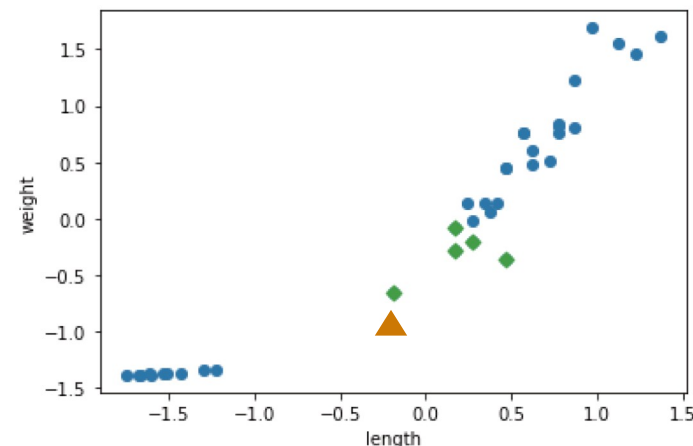
```
print(kn.predict([new]))
```

→ [1.]

- kneighbors() 함수로 이 샘플의 k-최근접 이웃을 구한 다음 산점도 그리기

```
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()
```

→



SECTION 2-2 데이터 전처리(15)

- 스케일이 다른 특성 처리(문제해결 과정)

- 문제

- 도미에 가까운 샘플(25cm, 150g)을 빙어라고 엉뚱한 예측이 발생
- 샘플의 두 특성인 길이와 무게의 스케일이 다르기 때문에 길이보다 무게의 크기에 따라 예측값이 좌우됨

- 해결

- 특성을 표준점수로 변환
- 특성의 스케일을 조정하는 방법은 표준점수 말고도 더 있지만 대부분의 경우 표준점수로 충분하며 가장 널리 사용하는 방법
- 주의: 데이터를 전처리할 때 훈련 세트를 변환한 방식 그대로 테스트 세트를 변환해야 함
 - 그렇지 않으면 특성값이 엉뚱하게 변환될 것이고 훈련 세트로 훈련한 모델이 제대로 동작하지 않음

SECTION 2-2 마무리

- 키워드로 끝나는 핵심 포인트
 - 데이터 전처리는 머신러닝 모델에 훈련 데이터를 주입하기 전에 가공하는 단계
 - 때때로 데이터 전처리에 많은 시간이 소모되기도 함
 - 표준점수는 훈련 세트의 스케일을 바꾸는 대표적인 방법 중 하나
 - 표준점수를 얻으려면 특성의 평균을 빼고 표준편차로 나눔
 - 반드시 훈련 세트의 평균과 표준편차로 테스트 세트를 바꿔야 함
 - 브로드캐스팅은 크기가 다른 넘파이 배열에서 자동으로 사칙 연산을 모든 행이나 열로 확장하여 수행하는 기능
- 핵심 패키지와 함수
 - scikit-learn
 - `train_test_split()`: 훈련 데이터를 훈련 세트와 테스트 세트로 나누는 함수
 - `kneighbors()`: k-최근접 이웃 객체의 메서드

SECTION 2-2 확인 문제

- 1 이 방식은 스케일 조정 방식의 하나로 특성값을 0에서 표준편차의 몇 배수만큼 떨어져 있는 지로 변환한 값임. 이 값을 무엇이라 부르나?

① 기본 점수	② 원점수
③ 표준점수	④ 사분위수
- 2 테스트 세트의 스케일을 조정하려고 합니다. 다음 중 어떤 데이터의 통계 값을 사용해야 하나?

① 훈련 세트	② 테스트 세트
③ 전체 데이터	④ 상관없음

SECTION 2-2 확인 문제

3. for 반복문을 사용하지 않고 넘파이 배열의 모든 원소에 대해 산술 연산이 적용되는 기능이 무엇인가요?

- ① 자동캐스팅 ② 연산캐스팅
- ③ 브로드캐스팅 ④ 브로드웨이

4. 다음 중 넘파이 배열 함수가 아닌 것은 무엇인가요?

- ① ones() ② nils()
- ③ mean() ④ std()