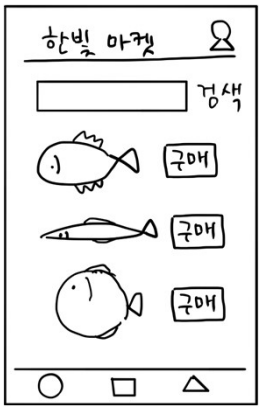
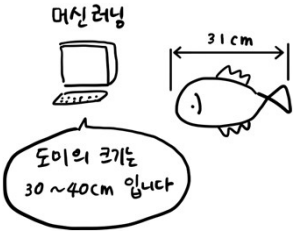
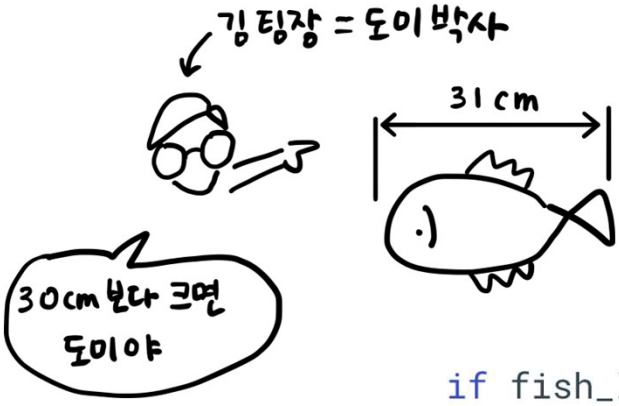
이번에 만들 머신러닝 프로그램





1

전통적인 프로그램




```
if fish_length >= 30:
    print("도미")
```

2

도미 vs 빙어

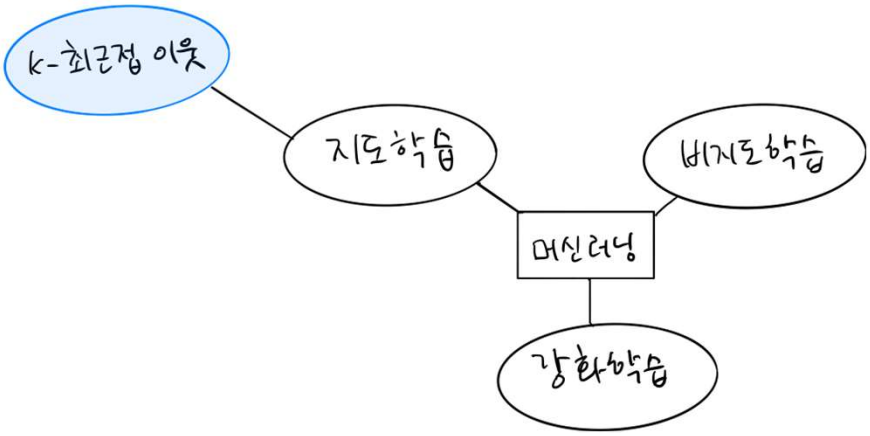
2개의 클래스(class)

분류(classification)

이진 분류(binary classification)

3

지도 학습과 비지도 학습

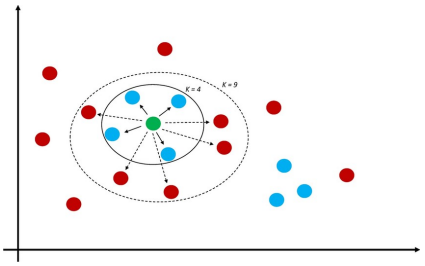


4

K-최근접 이웃 알고리즘

- 어떤 데이터에 대한 답을 구할 때 주위의 다른 데이터를 보고 다수를 차지하는 것을 정답으로 사용

- 마치 근묵자흑과 같이 주위의 데이터로 현재 데이터를 판단 함



- 이것을 위해 준비해야 할 일은 데이터를 모두 가지고 있어야 함

5

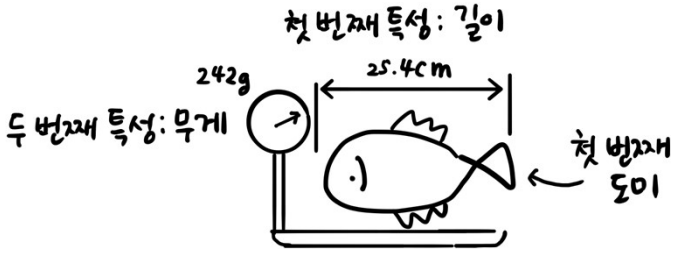
K-최근접 이웃 알고리즘

- **장점**
  - 새로운 데이터를 예측할 때는 가장 가까운 직선거리에 어떤 데이터가 있는지를 살피기만 하면 됨

- **단점**
  - 이 알고리즘의 이런 특징 때문에 데이터가 아주 많은 경우 사용하기 어려움
  - 데이터 크기 때문에 메모리가 많이 필요하고 직선거리를 계산하는 데도 많은 시간 필요

6

도미 데이터



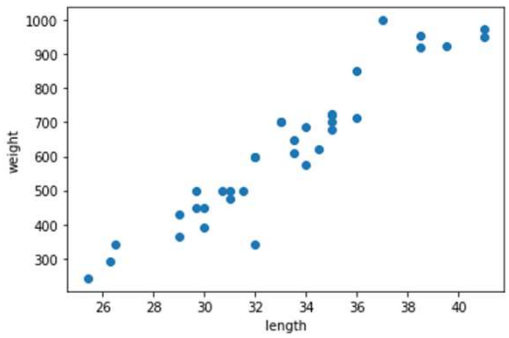
도미 35 마리 length 길이(cm)와 weight 무게(g)를 가진 특성(Feature) 데이터

```
bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7,
                31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5,
                34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0,
                38.5, 38.5, 39.5, 41.0, 41.0]
bream_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0,
                450.0, 500.0, 475.0, 500.0, 500.0, 340.0, 600.0, 600.0,
                700.0, 700.0, 610.0, 650.0, 575.0, 685.0, 620.0, 680.0,
                700.0, 725.0, 720.0, 714.0, 850.0, 1000.0, 920.0, 955.0,
                925.0, 975.0, 950.0]
```

7

산점도(scatter plot)

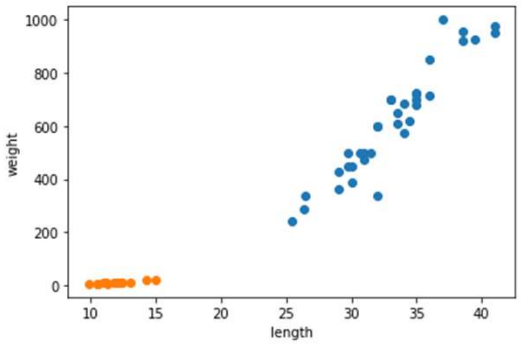
```
import matplotlib.pyplot as plt

plt.scatter(bream_length, bream_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



8

빙어 데이터

```
smelt_length = [9.8, 10.5, 10.6, 11.0, 11.2, 11.3, 11.8, 11.8, 12.0, 12.2,
                12.4, 13.0, 14.3, 15.0]
smelt_weight = [6.7, 7.5, 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4,
                12.2, 19.7, 19.9]

plt.scatter(bream_length, bream_weight)
plt.scatter(smelt_length, smelt_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



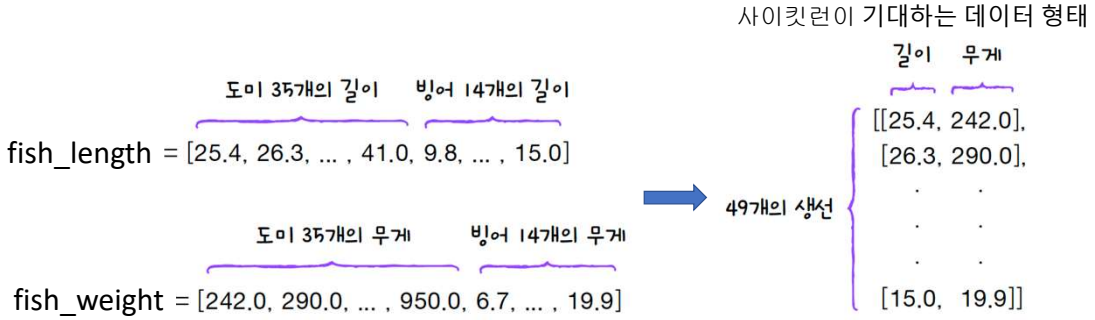
빙어 14 마리 length 길이(cm) 와 weight 무게(g) 를 가진 특성(Feature) 데이터

9

도미와 빙어 합치기

```
fish_length = bream_length + smelt_length
fish_weight = bream_weight + smelt_weight
```

사이킷런이 기대하는 데이터 형태

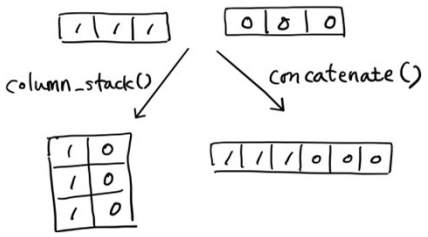


10

넘파이로 데이터 준비

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

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



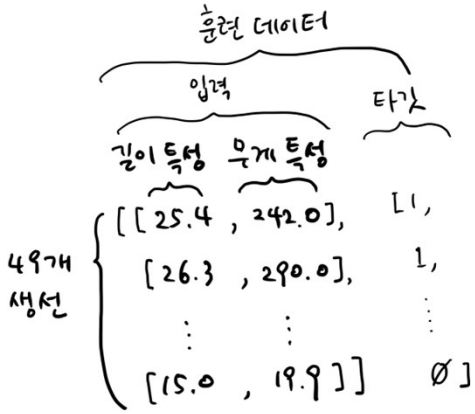

```
fish_target = np.concatenate((np.ones(35), np.zeros(14)))
```

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

정답 준비(데이터 라벨링) 1 :도미
0 :빙어

11

넘파이로 데이터 준비



12

사이킷런으로 데이터 나누기

지도학습에는 학습데이터와 정답이 필요. 이것을 트레이닝 데이터 Training Data 라고 함
- input : 학습데이터
- target : 정답

지도학습은 정답(target)이 있으니 알고리즘이 정답을 맞히는 것을 학습함
- 여기서는 도미와 빙어 구분

머신러닝 알고리즘의 성능을 제대로 평가하려면 훈련 데이터와 평가에 사용할 데이터가 각각 달라야 함

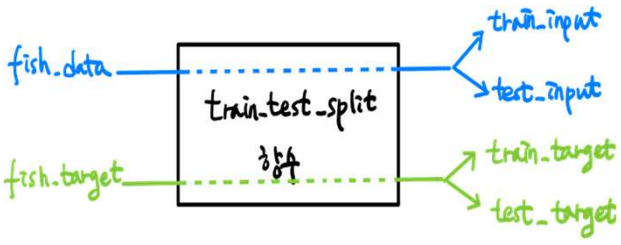
평가를 위해 또 다른 데이터를 준비하거나 이미 준비된 데이터 중에서 일부를 떼어 내어 활용함

13

사이킷런으로 데이터 나누기

샘플링 편향 Sampling bais (데이터가 한쪽으로 치우침)을 방지하기 위해 훈련세트와 테스트세트를 골고루 섞어야 함

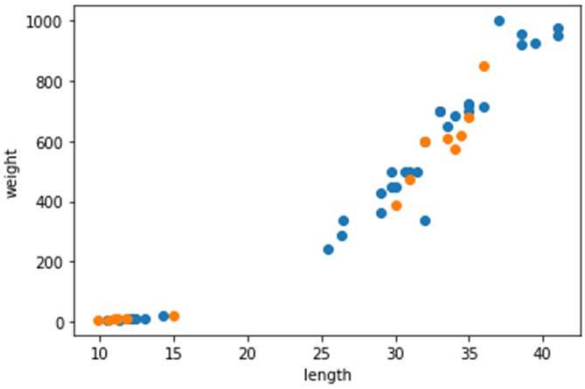
```
from sklearn.model_selection import train_test_split

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



14

데이터 나누고 확인하기

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

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()
```



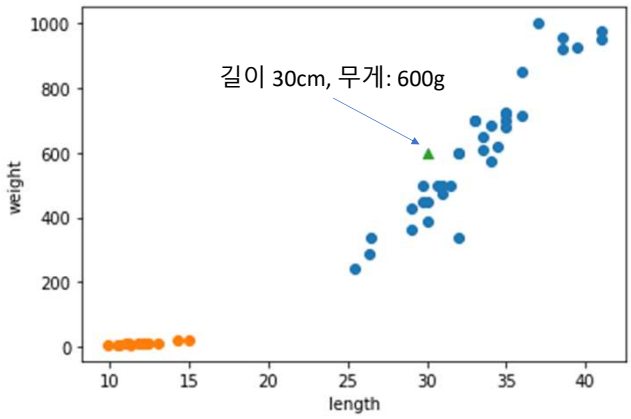
15

k-최근접 이웃

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier()

kn.fit(fish_data, fish_target)      #학습

kn.score(fish_data, fish_target)    #정확도 확인
1.0
```

16

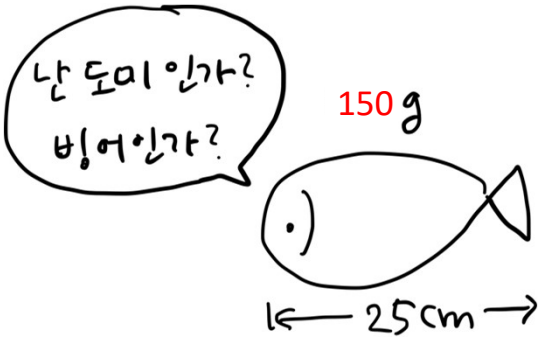
새로운 생선 예측




```
kn.predict([[30, 600]])

array([1])
```

17

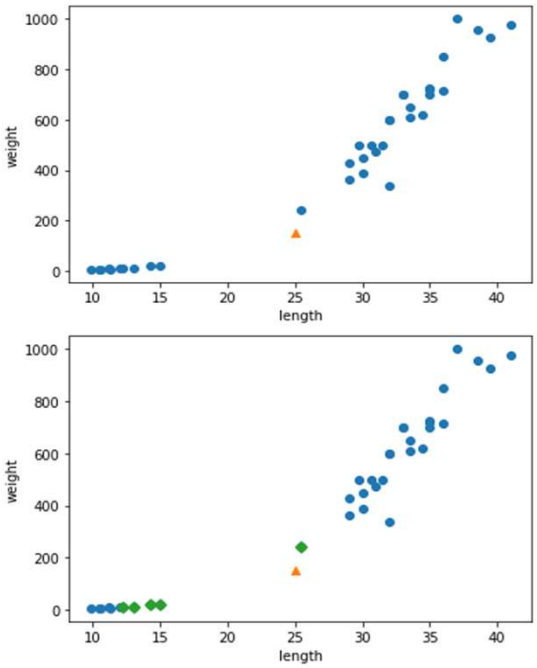
나는 누구인가?



18

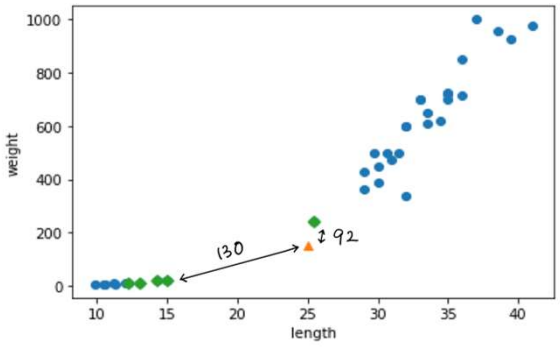
수상한 도미

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier()
kn.fit(train_input, train_target)
kn.score(test_input, test_target)
1.0

print(kn.predict([[25, 150]]))
[0.]

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

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()
```



19

기준을 맞춰라



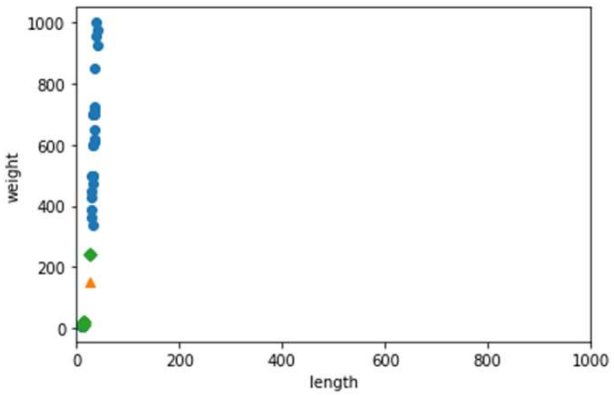
-x축의 범위는 좁고(10~40)
-y축의 범위는 넓음(0~1000)
- 그래서 y축으로 조금만 멀어져도 거리가 아주 큰 값으로 계산 됨
- 이 때문에 오른쪽 위의 도미 샘플이 이웃으로 선택되지 못함

20

기준을 맞춰라

명확하게 하기 위해 x축의 범위를 동일하게 (0~1000)으로 맞추어 분석

```
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()
```



21

기준을 맞춰라

- 산점도 그래프 결과, 거의 일직선으로 나타남
- 이는 두 특성(길이, 무게)의 값이 놓인 범위가 매우 다름
  : 두 특성의 Scale 차이가 남
- 두 Scale 특성이 다른 것은 흔함.
  데이터를 표현하는 기준이 다르면 알고리즘이 올바르게 예측 어려움.
- 특히 거리기반 알고리즘일 때는 영향을 많이 받음.
  제대로 사용하려면 특성 값을 일정한 기준으로 맞춰 줘야 함.
- 이러한 작업을 데이터 전처리Data preprocessing 이라함

22

표준 점수로 바꾸기

- 데이터 전처리 할때 주의할 점은 훈련 세트를 변환한 방식 그대로 테스트 세트에 그대로 적용.
- 그렇지 않으면 특성 값이 엉뚱하게 변환되고, 훈련 세트로 훈련된 모델이 제대로 동작하지 않음.
- 여기선 **표준 점수**를 이용해 훈련세트 스케일을 변경함.

23

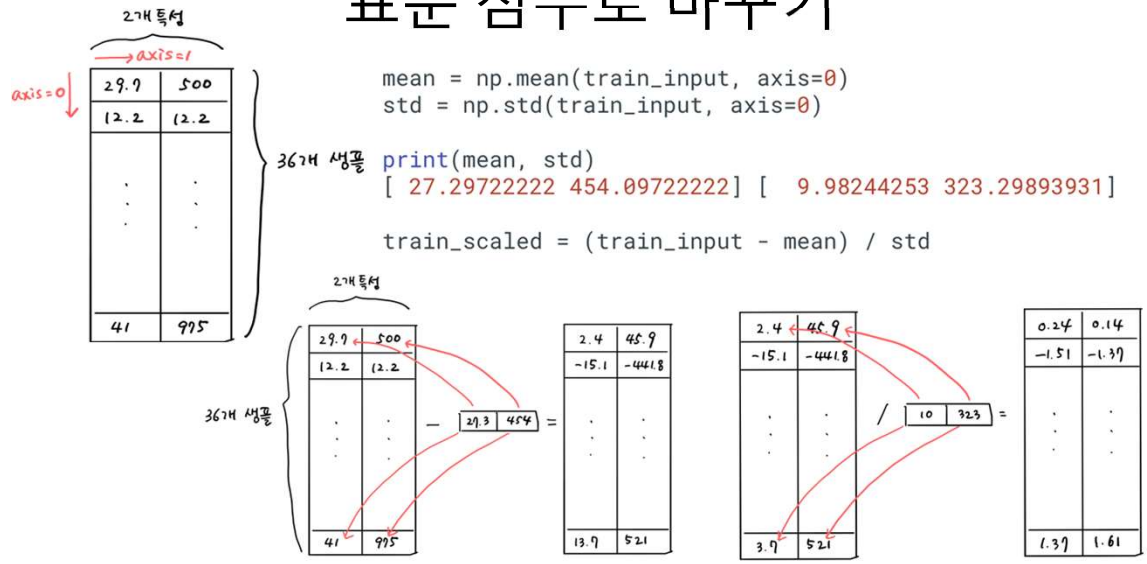
표준 점수로 바꾸기

- 특성마다 값의 스케일이 달라 평균과 표준편차는 각 특성별로 계산해야 함.
- 이를 위해 행을 따라 각 열의 통계 값을 계산하기 위해 axis=0로 지정함
- train_input의 모든 행에서 mean에 있는 두 평균값을 빼줌.
- 이후 std에 있는 두 표준편차를 다시 모든 행에 적용

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

print(mean, std)
[ 27.29722222 454.09722222] [  9.98244253 323.29893931]

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

24

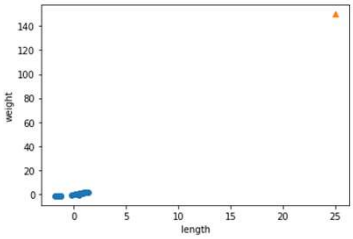
표준 점수로 바꾸기



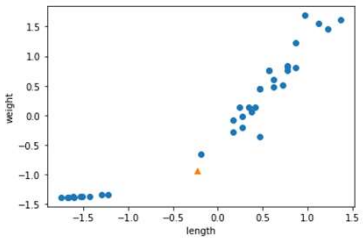

```
mean = np.mean(train_input, axis=0)
std = np.std(train_input, axis=0)

print(mean, std)
[ 27.29722222 454.09722222] [  9.98244253 323.29893931]

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

25

수상한 도미 다시 표시하기



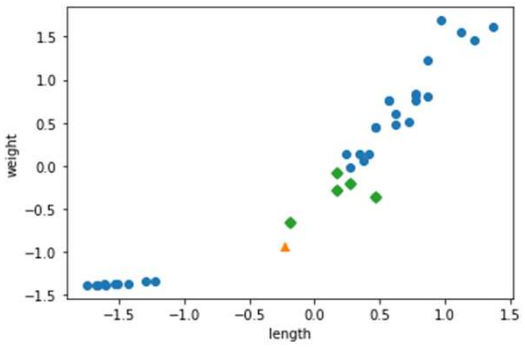

```
new = ([25, 150] - mean) / std

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



26

전처리 데이터에서 모델 훈련

```
kn.fit(train_scaled, train_target)

test_scaled = (test_input - mean) / std
kn.score(test_scaled, test_target)
1.0

print(kn.predict([new]))
[1.]

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()
```



27

감사합니다

내용 출처 정보 : https://www.hanbit.co.kr/store/books/look.php?p_code=B2002963743

28