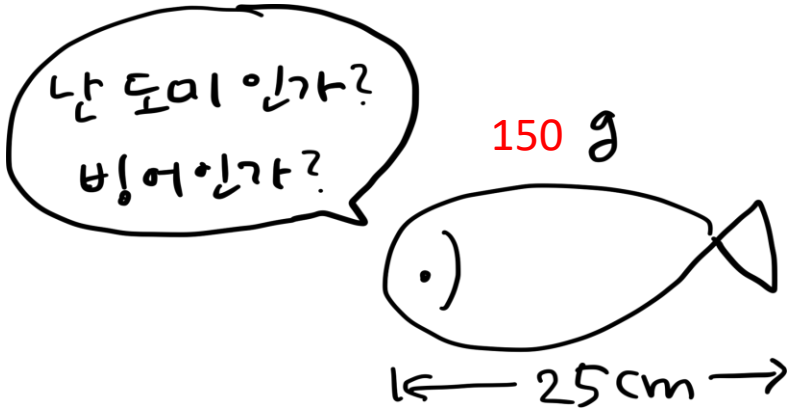
머신러닝 소개

- 지도학습알고리즘
- k-NN 분류와 회귀문제

새로운 데이터 분류에
k-NN 모델을 적용해도 괜찮은가?

넘파이와 사이킷런 라이브러리 활용



• 준비된 데이터

http://bit.ly/bream_smelt

도미 35개의 길이 빙어 14개의 길이

fish_length = [25.4, 26.3, ... , 41.0, 9.8, ... , 15.0]

도미 35개의 무게 빙어 14개의 무게

fish_weight = [242.0, 290.0, ... , 950.0, 6.7, ... , 19.9]

```
fish_data = [[l, w] for l, w in zip(length, weight)]
```

• 학습데이터 준비와 분할

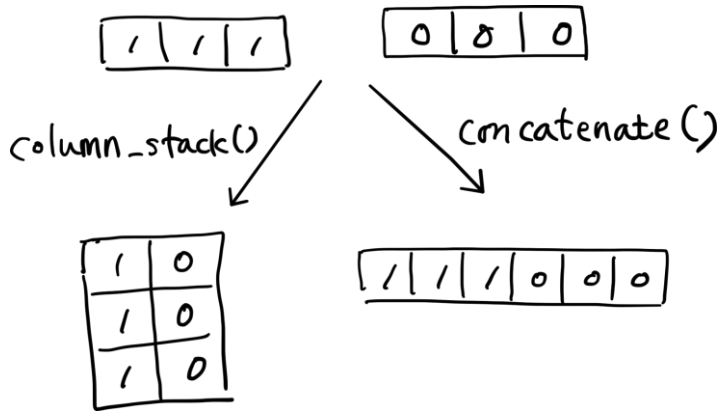
➢ 넘파이로 학습데이터 묶기

```
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)))
```

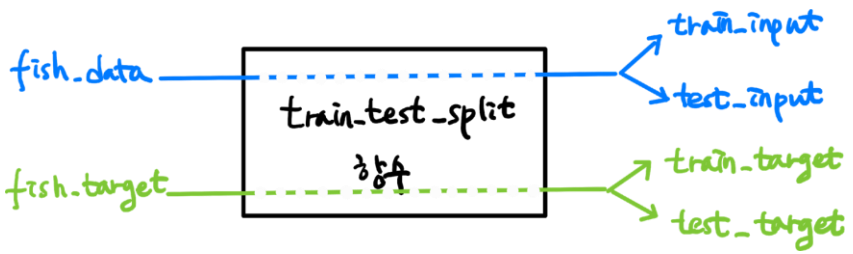
```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]
```



• 학습데이터 준비와 분할

➢ 사이킷런으로 데이터 나누기

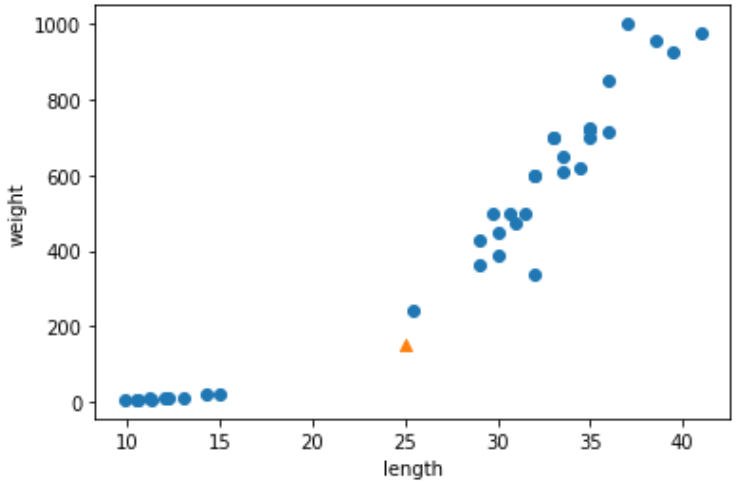
```
from sklearn.model_selection import train_test_split

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



• K-NN 모델로 새로운 데이터 예측

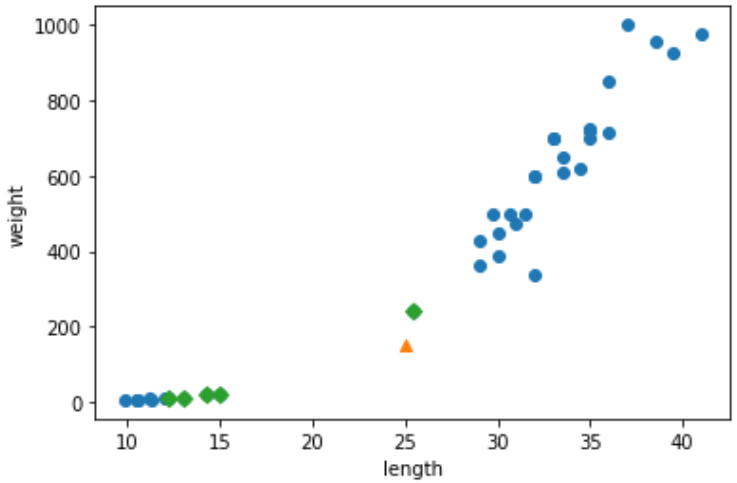
➢ 수상한 도미

```
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.] ← 빙어로 예측됨
```



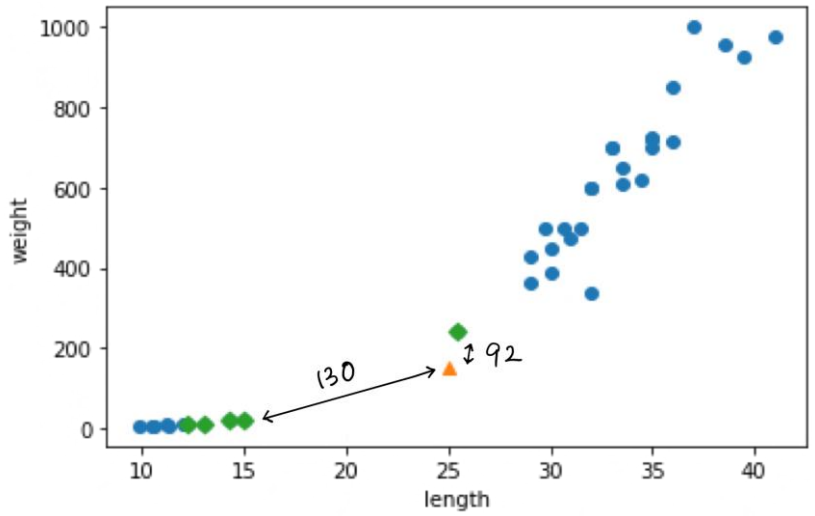

```
distances, indexes = kn.kneighbors([[25, 150]])

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()
```

이웃 샘플을 구분해 그리기



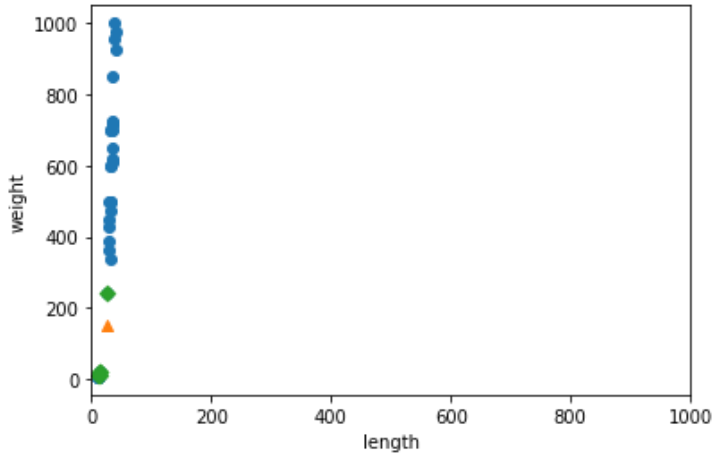
• K-NN 모델에서 이웃 측정 방법-DISTANCE

➢ 특성 단위(기준)가 같은지 고려하라.
length: cm , weight: g




```
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()
```

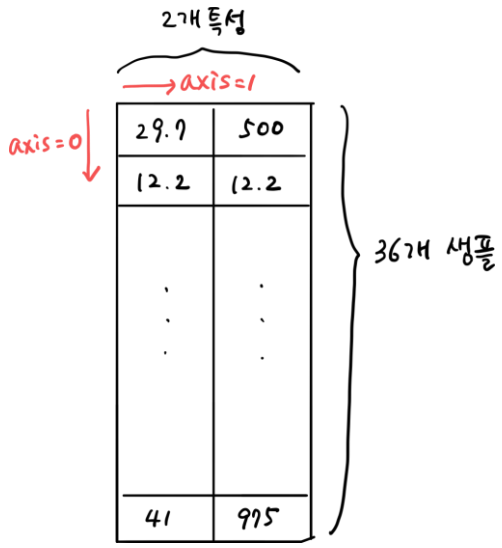
x축의 범위 지정



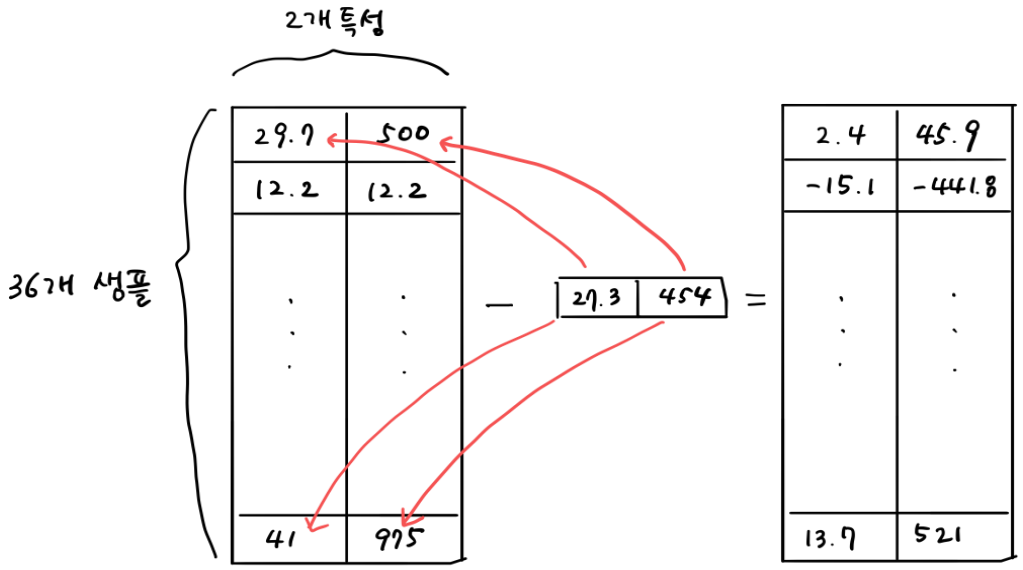
-> x축의 범위와 y축의 범위 비율 고려할 때

• 표준 점수로 바꾸기

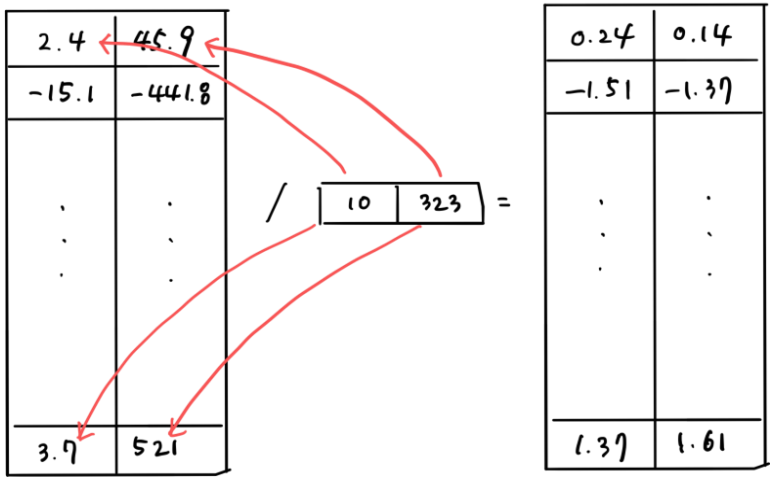
거리 기반 알고리즘을 다룰 때 주의 사항: 특성값을 일정 기준으로 맞추기->데이터 전처리
표준점수: 각 데이터가 원점에서 몇 표준편차만큼 떨어져 있는지를 나타내는 값




```
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
```

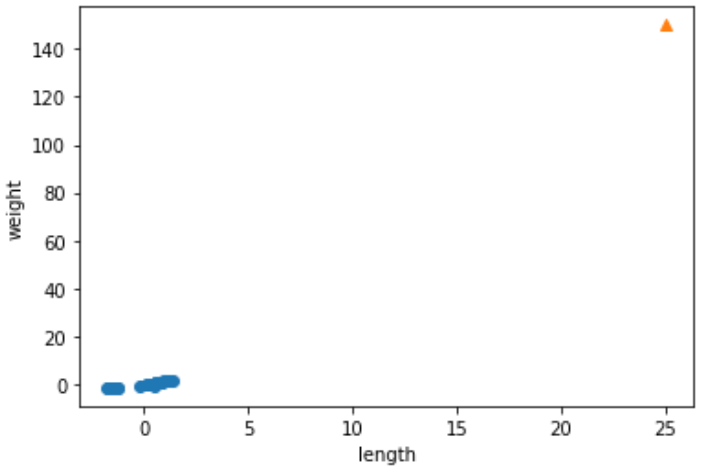


[평균 빼기]



[표준편차로 나누기]

• 표준점수로 데이터 전처리 후 수상한 도미 데이터 시각화



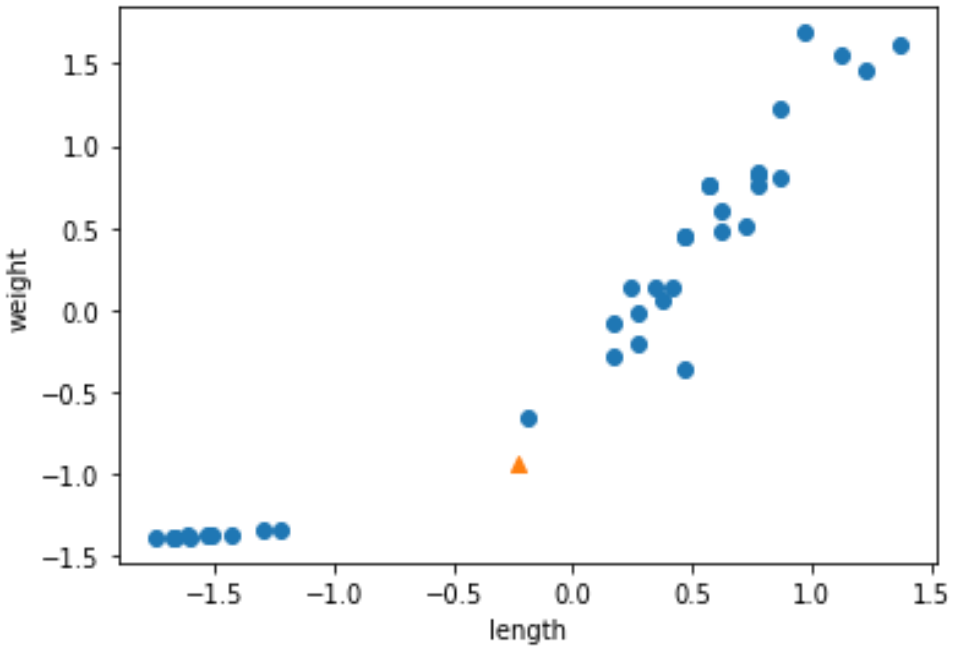

```
plt.scatter(train_scaled[:,0], train_scaled[:,1])
plt.scatter( 25,    150,      marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```

[전처리 데이터로 입력 데이터 예측 그리기]

=> 입력데이터도 훈련데이터와 같은 비율(표준점수화)로 변화해야만 한다.

• 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()
```



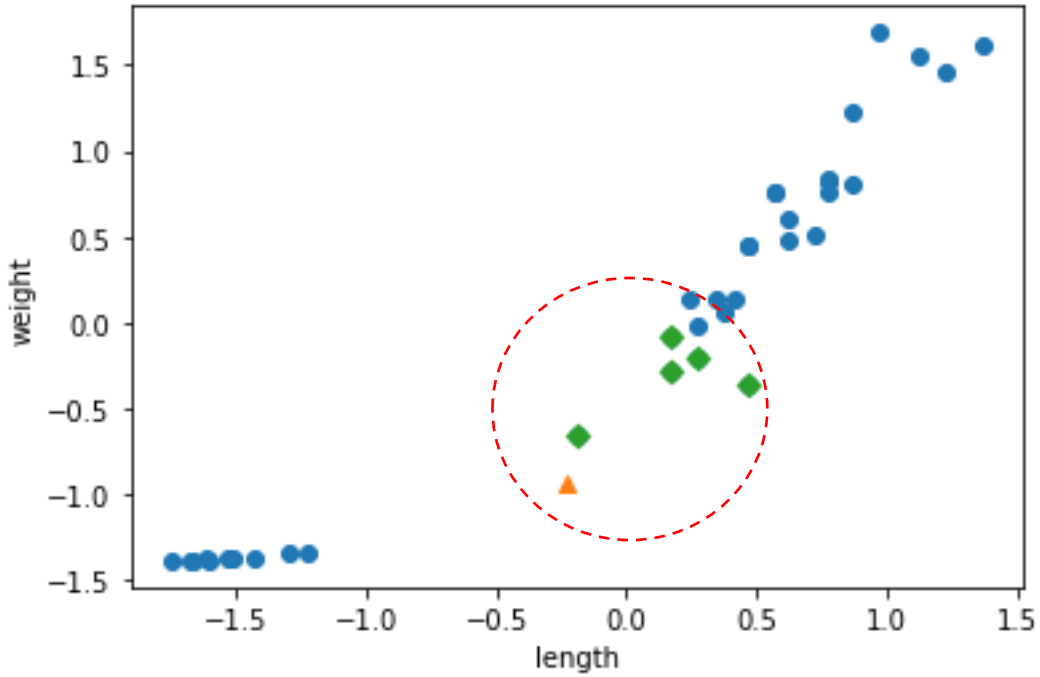
X축과 y축의 범위가 [-1.5~1.5]로 변경됨

• 전처리 데이터에서 모델 훈련

*** 훈련 데이터의 평균과 표준편차로 테스트 데이터도 표준점수화해야 한다.

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

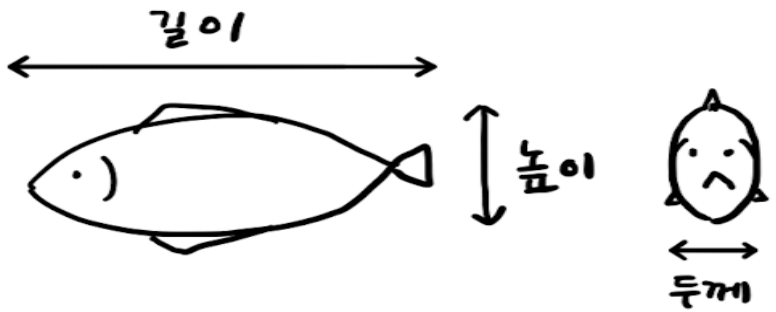
```
test_scaled = (test_input - mean) / std
kn.score(test_scaled, test_target)
1.0

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()
```



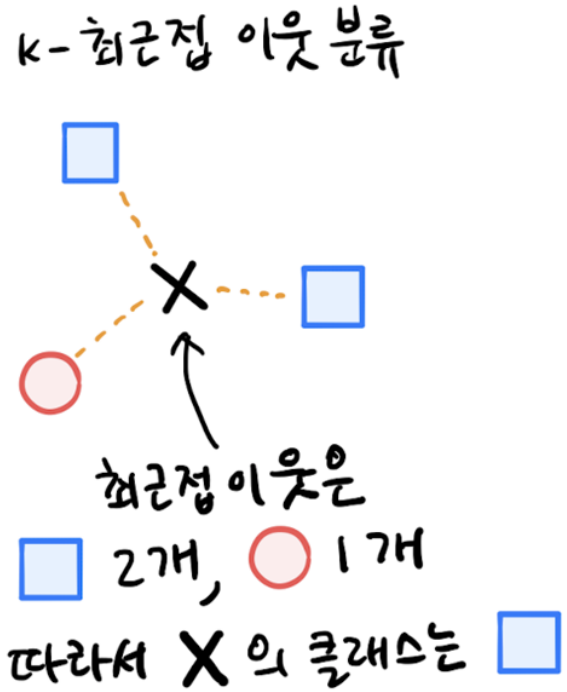
• k-NN 분류 알고리즘으로 무게 예측도 가능한가?

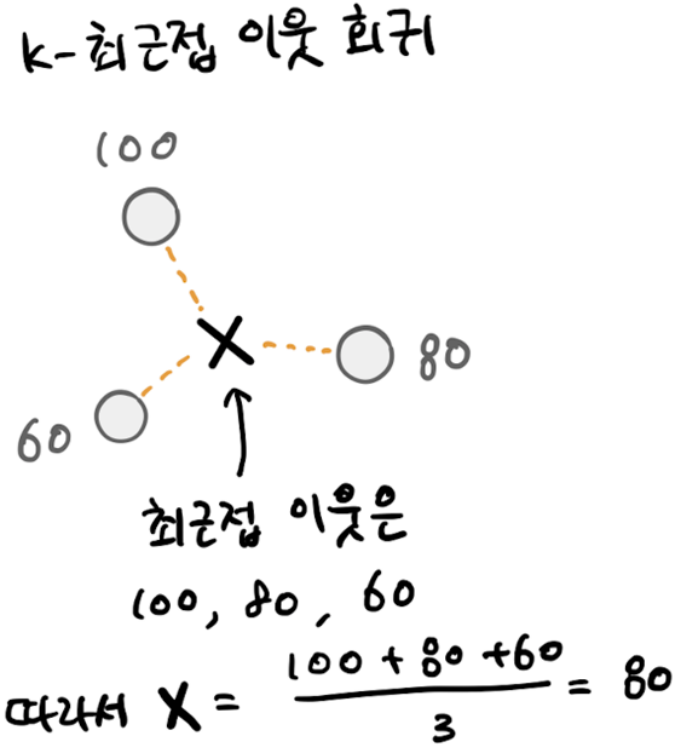
입력데이터와 가장 가까운 k개의 인접한 데이터 샘플 추출한 후
가장 많은 샘플데이터가 속한 클래스를 입력데이터의 클래스로 할당(분류)함

연속된 입력값에 대한 연속된 출력값을 예측하는 문제는 어떻게 해결할까?
예) 농어의 무게를 예측하라



• k-최근접 이웃 회귀

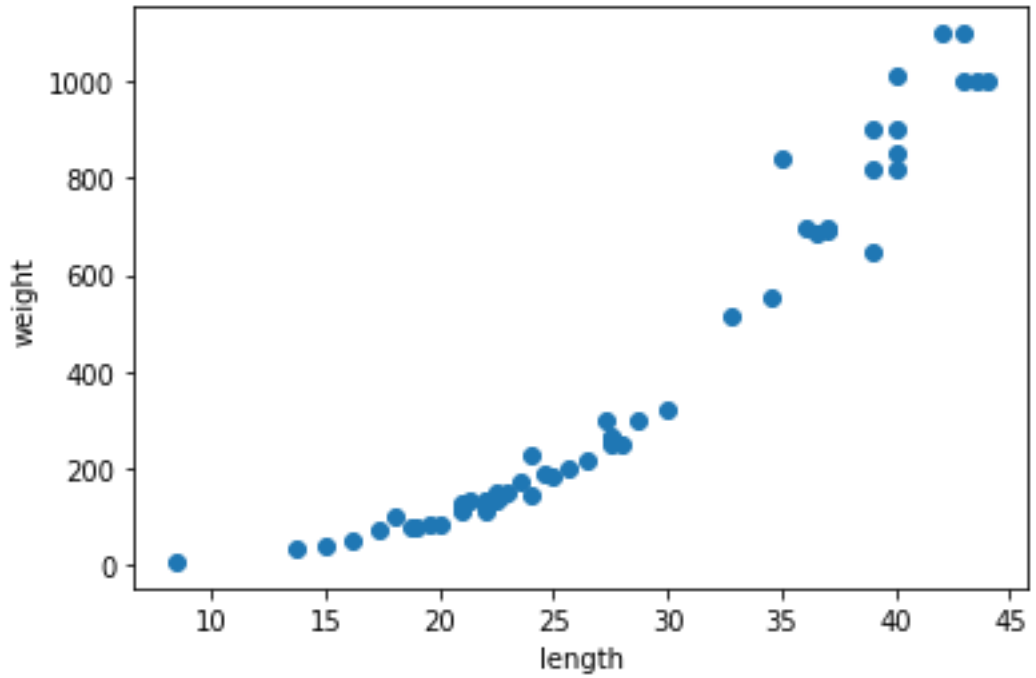




- Classification(분류): 데이터를 분류하는 가상 경계 찾기(범주형)
- Regression(회귀): 값을 찾는 방법(수치형)

• 농어의 길이, 무게 특성 데이터 준비

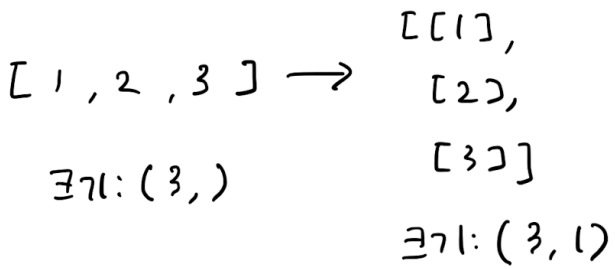
```
import matplotlib.pyplot as plt

plt.scatter(perch_length, perch_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



• 데이터 셋 나누기 train_test_split()

```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(
    perch_length, perch_weight, random_state=42)

train_input = train_input.reshape(-1, 1)
test_input = test_input.reshape(-1, 1)
```

->사이킷런에서 다루는 훈련세트는 2차원 배열이어야 한다.



• K-NN 회귀

```
from sklearn.neighbors import KNeighborsRegressor

knr = KNeighborsRegressor()
knr.fit(train_input, train_target)
```

회귀모델훈련

```
knr.score(test_input, test_target)
0.9928094061010639
```

회귀모델평가

$$R^2 = 1 - \frac{(\text{타깃} - \text{예측})^2 \text{의 합}}{(\text{타깃} - \text{평균})^2 \text{의 합}}$$

[결정계수 이용한 회귀모델 평가]

```
from sklearn.metrics import mean_absolute_error

test_prediction = knr.predict(test_input)
mae = mean_absolute_error(test_target, test_prediction)
print(mae)
19.157142857142862
```

테스트셋에 대한 평균 절대값 오차 계산

농어의 무게 예측값은 평균 19 g 정도 차이남

• 과대적합과 과소적합

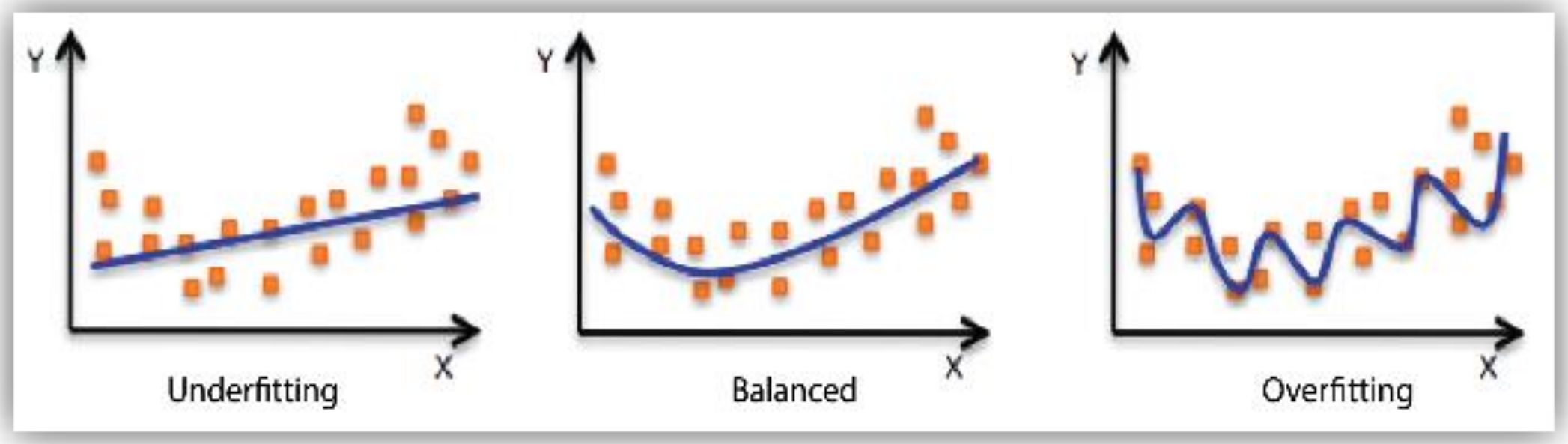
훈련세트로도 모델 평가, 테스트세트로 평가한 값과 비교

```
knr.score(train_input, train_target)
0.9698823289099255

knr.score(test_input, test_target)
0.9928094061010639
```

과소적합

훈련데이터셋보다 테스트셋 평가 값이 더 높게 나오거나 두개의 평가 값이 모두 낮을 경우 과소적합되었다고 함

• 과대적합/과소적합



과대적합(Overfitting) VS 과소적합(Underfitting)

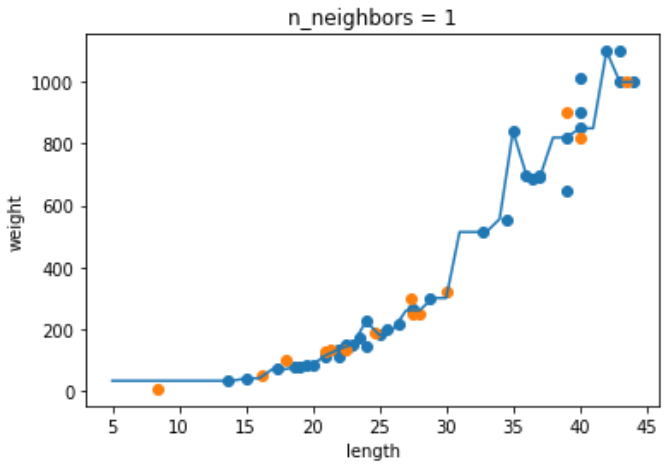
• 과대적합/과소적합

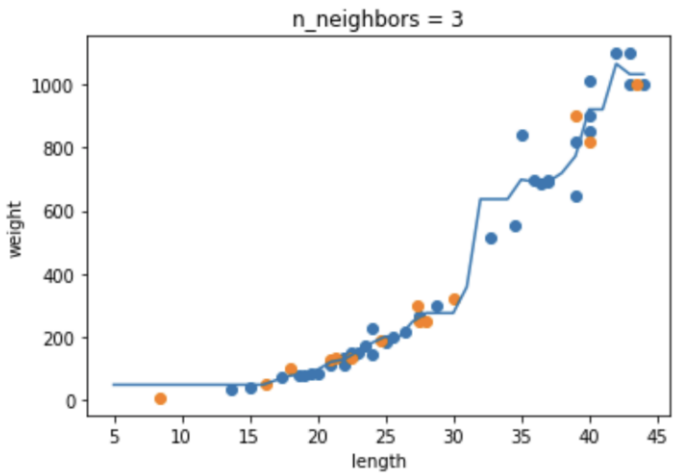
- **과대적합(Overfitting):** 모델이 훈련 데이터에 너무 잘 맞지만 일반성이 떨어짐
  - 훈련데이터는 실제로 존재하는 많은 데이터의 일부에 불과, 머신러닝 모델이 훈련데이터에 대해서만 과도하게 학습하면 진짜 실전 데이터나 서비스에서 정확도가 좋지 않은 현상이 발생한다는 의미
- **과소적합(Underfitting):** 과대적합의 반대. 모델이 너무 단순해서 데이터의 포함된 의미를 제대로 학습하지 못할 때 발생
  - 약간만 기울기를 조정하면 비용함수를 줄일 수 있는 모델임에도 불구하고 단순 직선 모델로 일반화에 부적합하게 예측된 경우.
- 머신러닝에서 데이터로 모델을 만드는 과정을 학습 또는 훈련이라고 함.
  - 주어진 데이터에 대해서 적당하게 조정한다는 의미에서 피팅(fitting)이라고 함
  - 머신러닝 모델의 fit() 함수

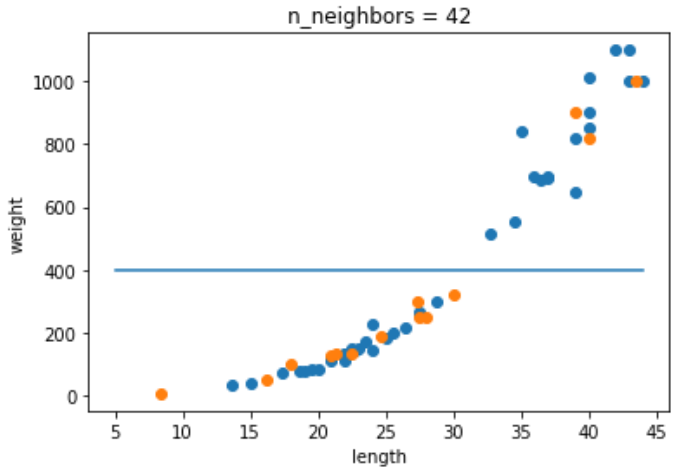
• K-NN 회귀의 과적합 문제 해결: k값 조절

```
knr.n_neighbors = 3
knr.fit(train_input, train_target)

print(knr.score(train_input, train_target))
0.9804899950518966

print(knr.score(test_input, test_target))
0.974645996398761
```

과대적합 ↓ 이웃의 개수 ↑ 과소적합







K-NN 회귀의 문제점

k-NN 회귀로 농어 무게 예측?

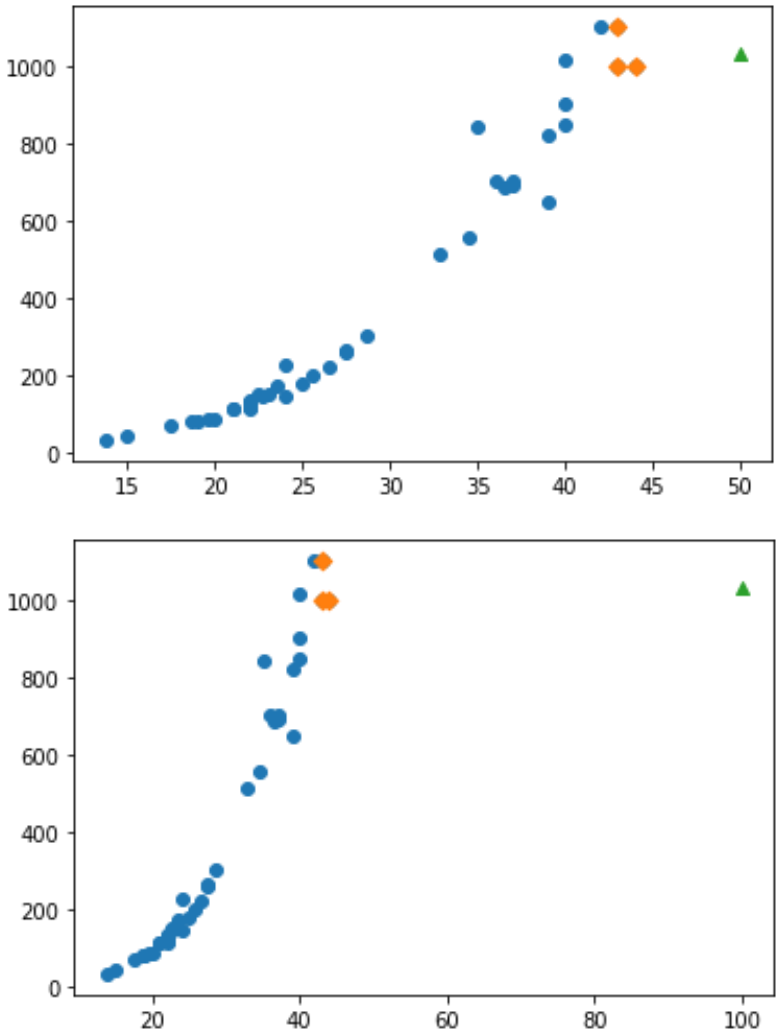
KNN 방식으로 새로운 데이터를 예측할 때 발생할 수 있는 문제는 무엇일까?
50cm 농어의 무게와 100cm 농어의 무게는 같을까? 다를까?


```
print(knr.predict([[50]]))
[1033.33333333]
```

k-NN 회귀로 농어 무게 예측?

50cm 농어의 이웃은?

```
# 50cm 농어의 이웃을 구합니다
distances, indexes = knr.kneighbors([[50]])

# 훈련 세트의 산점도를 그립니다
plt.scatter(train_input, train_target)
# 훈련 세트 중에서 이웃 샘플만 다시 그립니다
plt.scatter(train_input[indexes], train_target[indexes],
            marker='D')
# 50cm 농어 데이터
plt.scatter(50, 1033, marker='^')
plt.show()
```



*** 새로운 샘플이 훈련 세트의 범위를 벗어나면 엉뚱한 값 예측할 수 있다.