Data Intelligence Lab

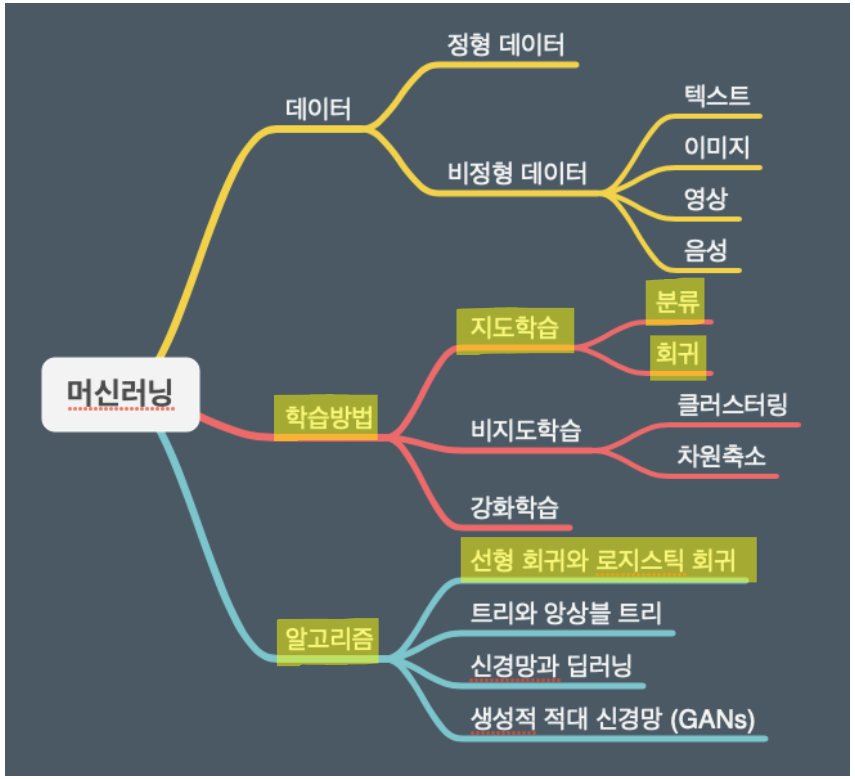
Progress

6월 22 ~ 29

인천대학교 컴퓨터공학부 강병하

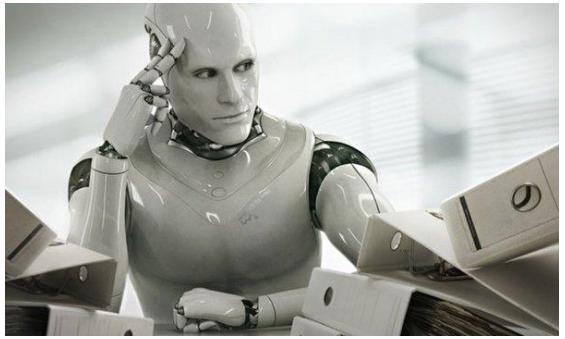
무엇을 공부했는가


혼자 공부하는 머신러닝 +딥러닝
인공지능 공부가 정말 처음일 때
어려운 수식에 지쳤을 때
쉬운 그림과 실전 예제로 공부하고 싶을 때
박해선 지음
★★★★★
신 혼공머신
혼자 공부하는 일에 능숙한 사람 혹은 그런 무리를 일컫는 신조어
(유의어) 혼공족, 혼공파, 혼공자, 혼공씨
구글 코랩으로 환경 설정 없이 실습 가능
유튜브 강의
별책 용어 노트
TALK
저자와 함께하는 오픈채팅
http://bit.ly/tensor-chat
한빛미디어


머신러닝
데이터
정형 데이터
비정형 데이터
텍스트
이미지
영상
음성
학습방법
지도학습
분류
회귀
비지도학습
클러스터링
차원축소
강화학습
알고리즘
선형 회귀와 로지스틱 회귀
트리와 앙상블 트리
신경망과 딥러닝
생성적 적대 신경망 (GANs)

세 가지 모델 실습

1. 빙어인지 도미인지
   판별하는 **이진분류 모델**



2. 농어의 무게를 예측하는
   회귀 모델

3. 생성의 종을 판별하는
   로지스틱 회귀 모델

첫 번째 모델 : 빙어와 도미의 이진분류

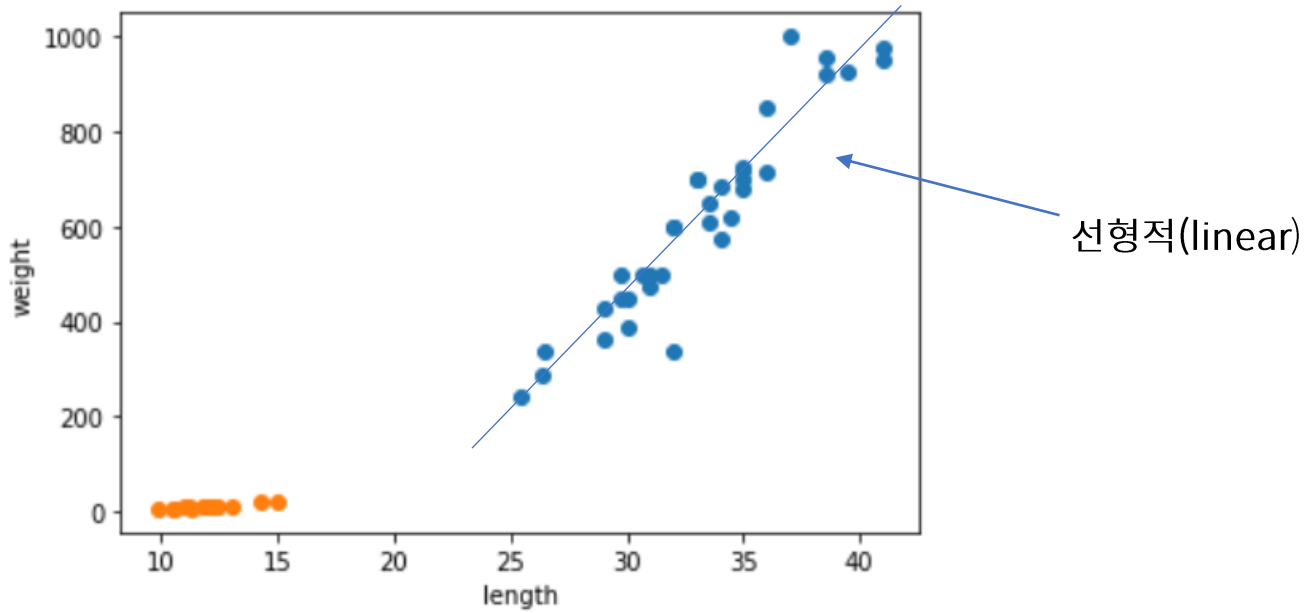
```
bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7, 31.0, 31.0,
                31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0,
                35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5, 39.5, 41.0, 41.0]
bream_weight = [242.0, 290.0, 340.0, 363.0, 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]
```

특성(feature) : 데이터의 특징

```
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]
```

첫 번째 모델 : 빙어와 도미의 이진분류 – 데이터 시각화

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

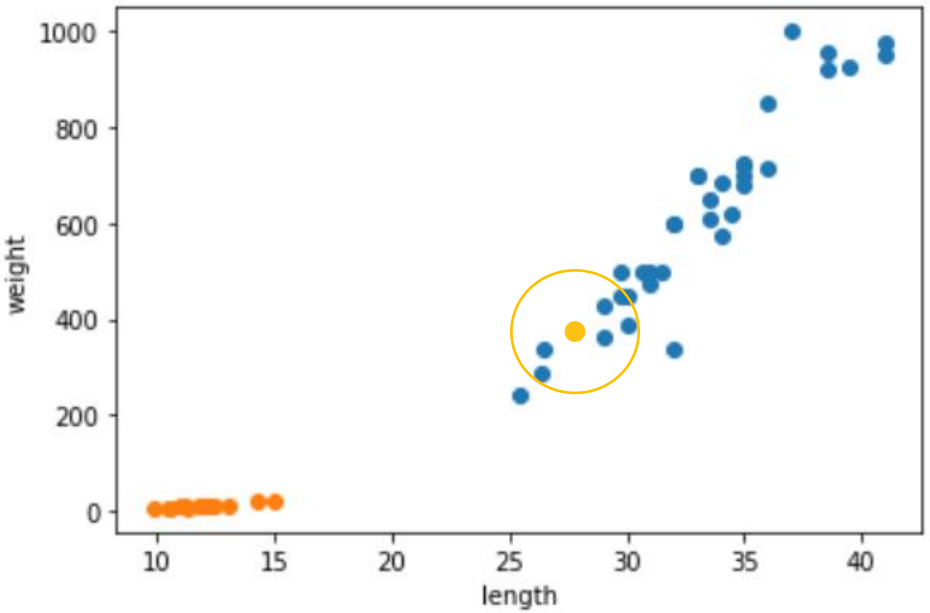


선형적(linear)

파란색 : 도미 주황색 : 빙어

첫 번째 모델 : 빙어와 도미의 이진분류 – k 최근접 이웃 알고리즘

k 최근접 이웃 알고리즘 -> 주위의 다른 데이터(기본값 5개)를 보고 다수를 차지하는 것으로 판단



“근접한 5개 모두 도미이므로
28cm 400g 생선은 빙어가 아니라 도미일 것”

첫 번째 모델 : 빙어와 도미의 이진분류 – 데이터 준비

```
length = bream_length + smelt_length
weight = bream_weight + smelt_weight
```

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

행렬 형태로 변환 (이중리스트)

도미는 1, 빙어는 0이라고 가정한다.

```
fish_target = [1] * 35 + [0] * 14
print(fish_target)
```

```
[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]
```

정답은 벡터 형태

첫 번째 모델 : 빙어와 도미의 이진분류 – 훈련 및 평가

```
from sklearn.neighbors import KNeighborsClassifier
```

```
kn = KNeighborsClassifier()
```

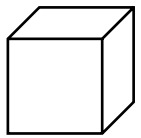
```
kn.fit(fish_data, fish_target)
```

지도학습에서는 입력(input)과 정답(target)을 전달.

```
kn.score(fish_data, fish_target)
```

```
1.0
```

정답을 외우고 시험을 치른 꼴



첫 번째 모델 : 빙어와 도미의 이진분류 – 훈련 세트, 테스트 세트

훈련에 사용하는 데이터와 평가에 사용하는 데이터는 달라야 한다.

-> 준비된 데이터에서 일부를 떼어냄

```
# 훈련 데이터
train_input = fish_data[:35]

# 훈련 타깃(정답)
train_target = fish_target[:35]

# 테스트 데이터
test_input = fish_data[35:]

# 테스트 타깃
test_target = fish_target[35:]
```

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

```
0.0
```

하나도 맞추지 못함 Why?

첫 번째 모델 : 빙어와 도미의 이진분류 – 샘플링 편향(sampling bias)

하나의 생선 데이터 -> **샘플(sample)** 총 49개의 샘플

```
fish_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, 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]
fish_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, 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]
```

훈련 세트 -> 도미 35마리, 테스트 세트 -> 빙어 14마리

샘플이 골고루 섞여 있지 않음 -> **샘플링 편향**.

과도한 샘플링 편향이 있다면 제대로 된 모델을 만들 수 없다.

첫 번째 모델 : 빙어와 도미의 이진분류 – 샘플링 편향 제거

```
import numpy as np
```

```
# 파이썬 리스트를 넘파이 배열로 바꾸기
input_arr = np.array(fish_data)
target_arr = np.array(fish_target)
```

```
np.random.seed(42) #시드값을 같게 주면 일정한 결과를 얻을 수 있다.
index = np.arange(49) # 0부터 48까지 1씩 증가하는 배열을 만들어준다. [0 1 2 ... 48]
np.random.shuffle(index) # 무작위로 섞는다.
```

```
# 훈련 세트 만들기
train_input = input_arr[index[:35]] # 랜덤하게 섞인 인덱스들로 하나씩 선택
train_target = target_arr[index[:35]]
```

```
# 테스트 세트 만들기
test_input = input_arr[index[35:]]
test_target = target_arr[index[35:]]
```

첫 번째 모델 : 빙어와 도미의 이진분류 – 샘플링 편향 제거

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

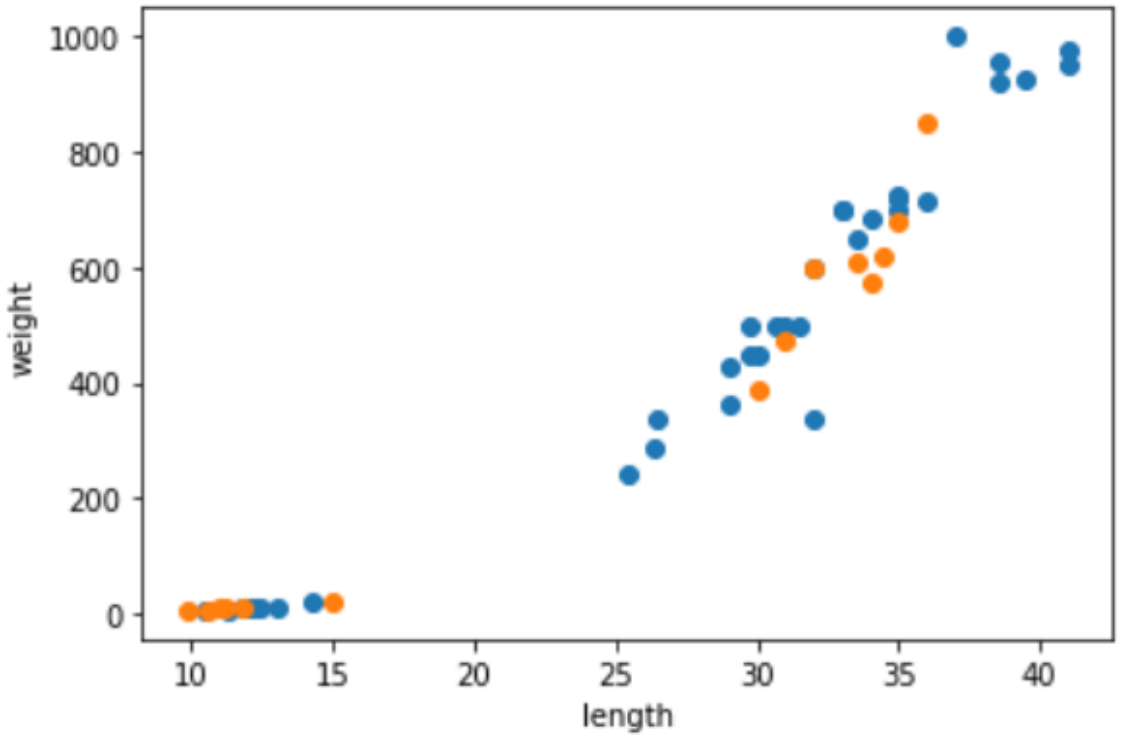
```
train_input, test_input, train_target, test_target = train_test_split(fish_data, fish_target, random_state=42)
```

사이킷런의 train_test_split() 메서드

알아서 훈련 세트와 테스트 세트를 분리 시켜줌

(전체에서 25%를 테스트 세트로)

첫 번째 모델 : 빙어와 도미의 이진분류 – 샘플링 편향 제거



파란색이 훈련 세트, 주황색이 테스트 세트이다.

훈련 세트와 테스트 세트가 골고루 섞여 있는 것을 볼 수 있다.

훈련 세트와 테스트 세트의
구성 비율이 비슷한 것이 좋다.

첫 번째 모델 : 빙어와 도미의 이진분류 – 재훈련 및 평가

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

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

```
1.0
```

```
kn.predict(test_input) # 넘파이 배열. 사이킷런 모델의
```

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

```
test_target
```

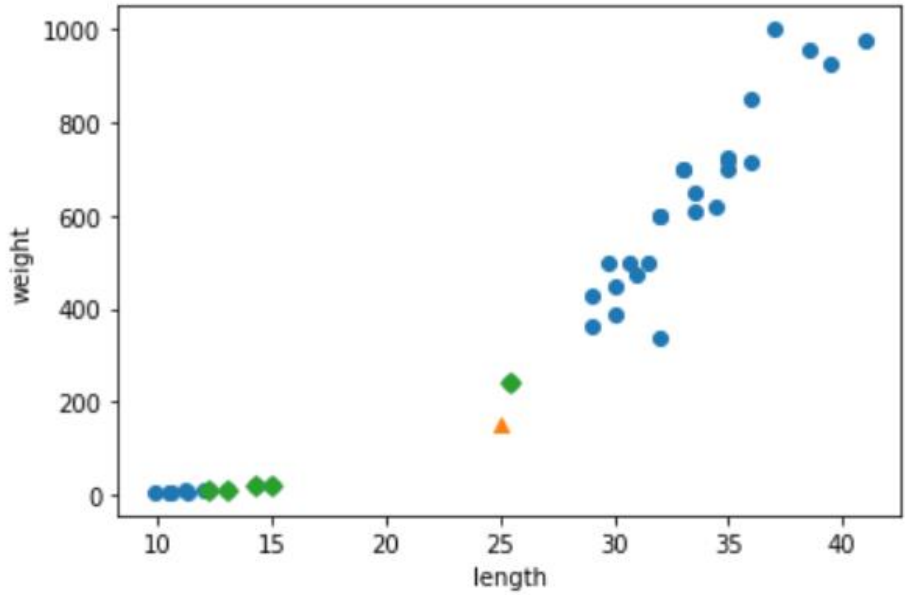
```
array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
```

첫 번째 모델 : 빙어와 도미의 이진분류 – 모델의 오류

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

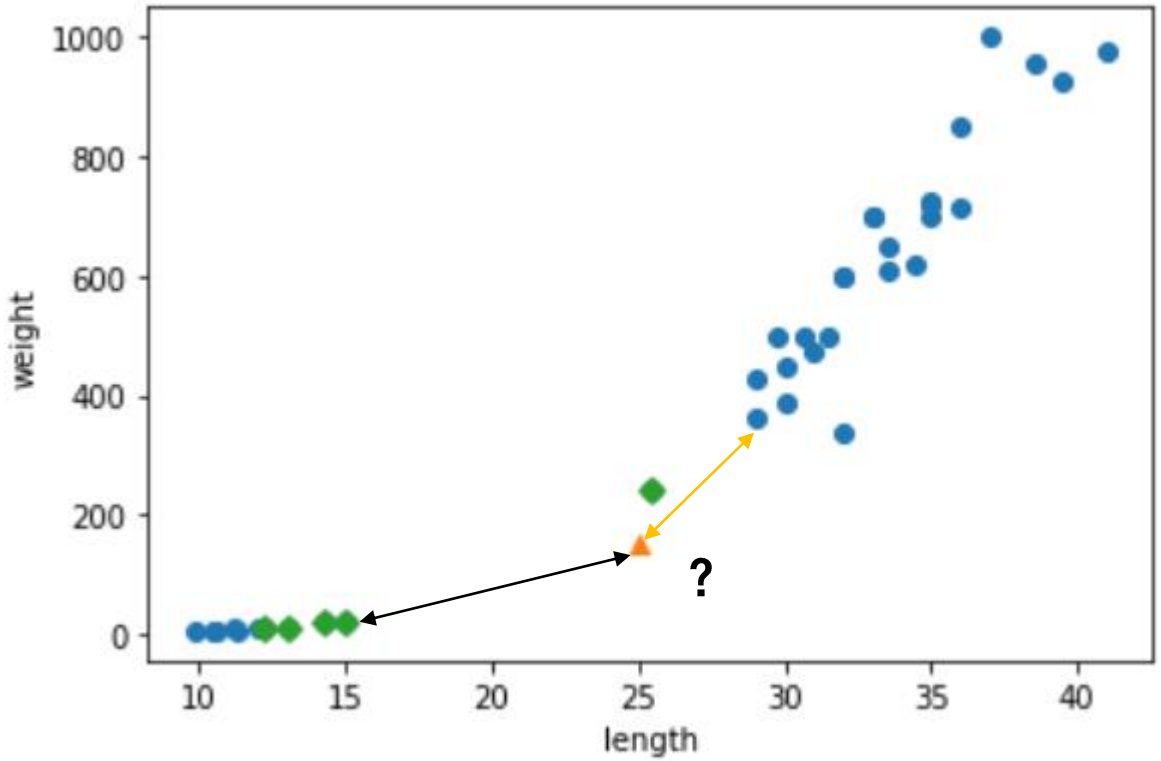
```
array([0])
```

1: 도미
0: 빙어



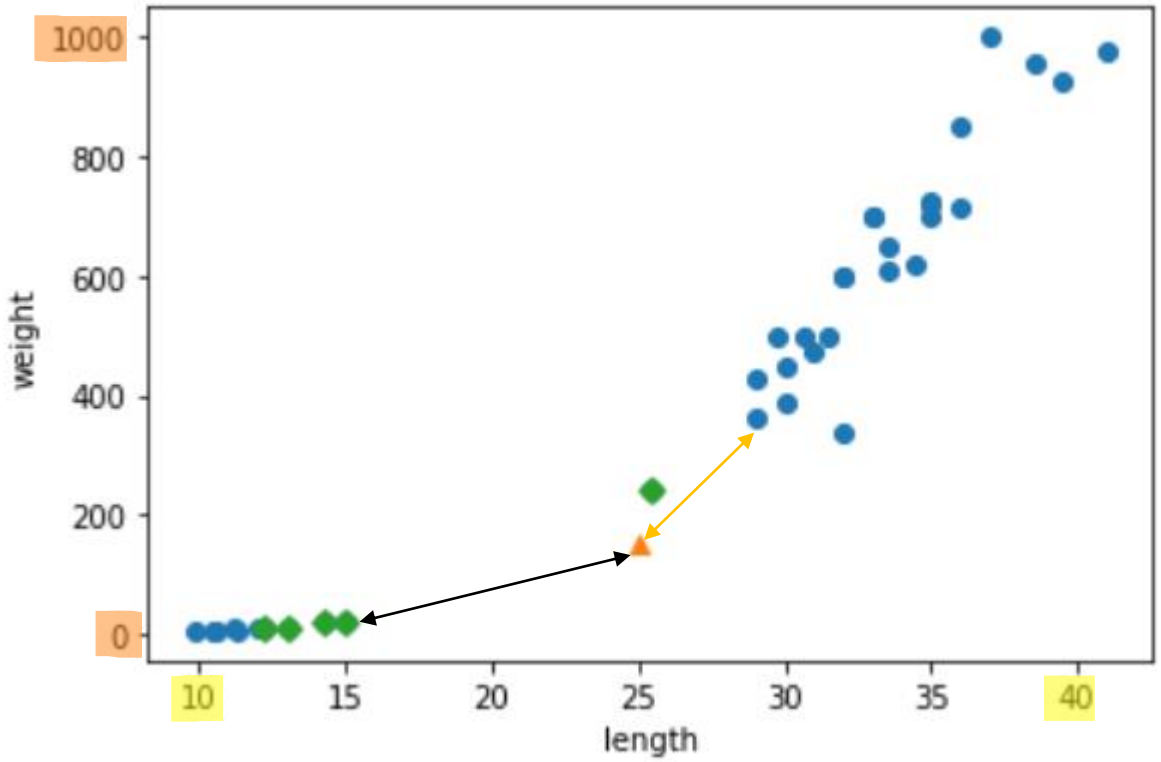
길이 25cm, 무게 150g의 도미를 빙어라 판단

첫 번째 모델 : 빙어와 도미의 이진분류 – 오류 원인 분석



첫 번째 모델 : 빙어와 도미의 이진분류 – 오류 원인 분석

weight : 0 ~ 1000



Length : 10 ~ 40

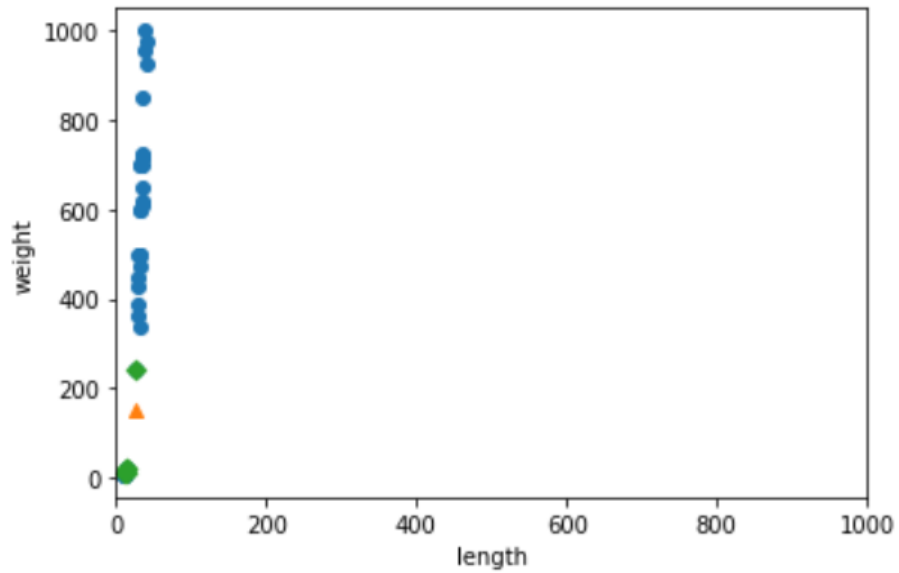
x축과 y축의 범위가 다르다

= 특성의 값의 범위가 다르다

= 스케일이 다르다

첫 번째 모델 : 빙어와 도미의 이진분류 – 오류 원인 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)) # 범위를 튜플로 입력
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



(특히 거리 기반 알고리즘의 경우)

기준이 다르다면 올바른 예측 불가능

-> 특성을 일정한 기준으로 맞춰야 한다.

첫 번째 모델 : 빙어와 도미의 이진분류 – 데이터 전처리(data preprocessing)

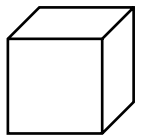
표준점수(standard score), z점수

평균으로부터
표준편차의 몇 배만큼 떨어져 있는가?

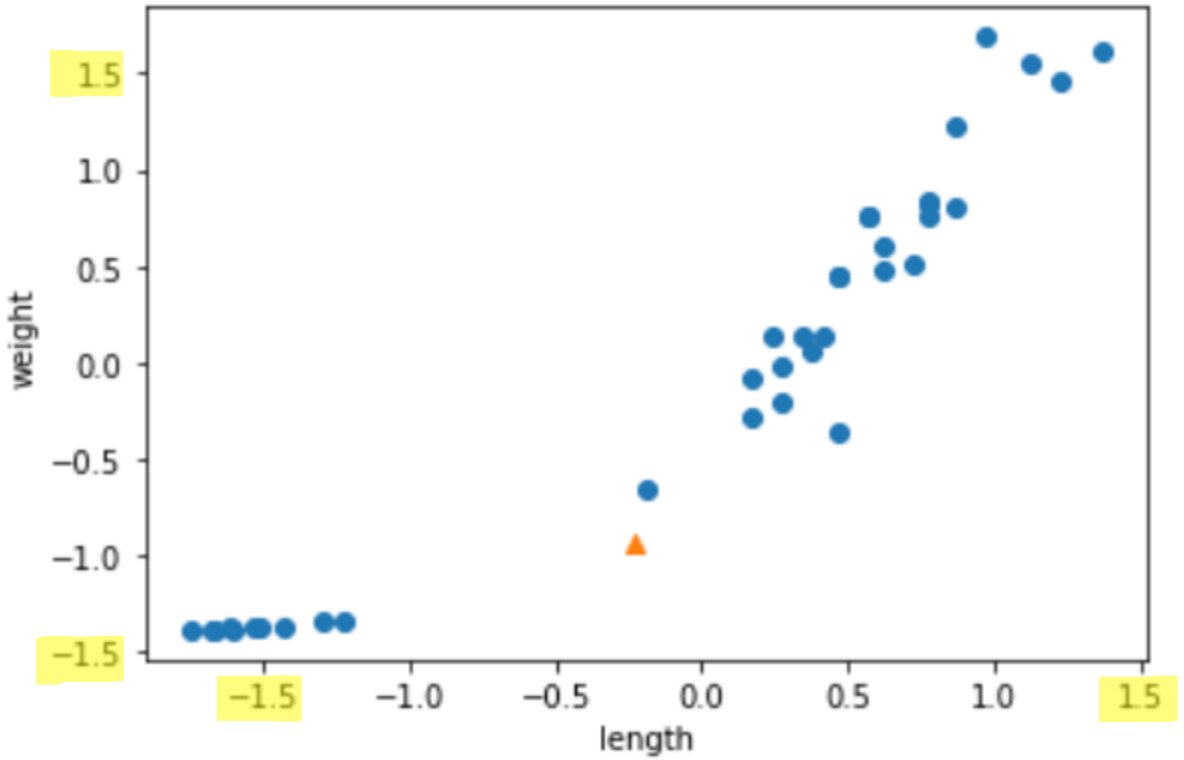
```
# 표준점수 구하기

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

* 표준편차 = 분산의 양의 제곱근
  분산(편차의 제곱의 평균)
전체적으로 얼마만큼 분산되어 있는가



첫 번째 모델 : 빙어와 도미의 이진분류 – 스케일 변환



첫 번째 모델 : 빙어와 도미의 이진분류 – 스케일 변환 후 재훈련

```
# 표준점수 구하기
train_scaled = (train_input - mean) / std
```

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

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

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

```
1.0
```

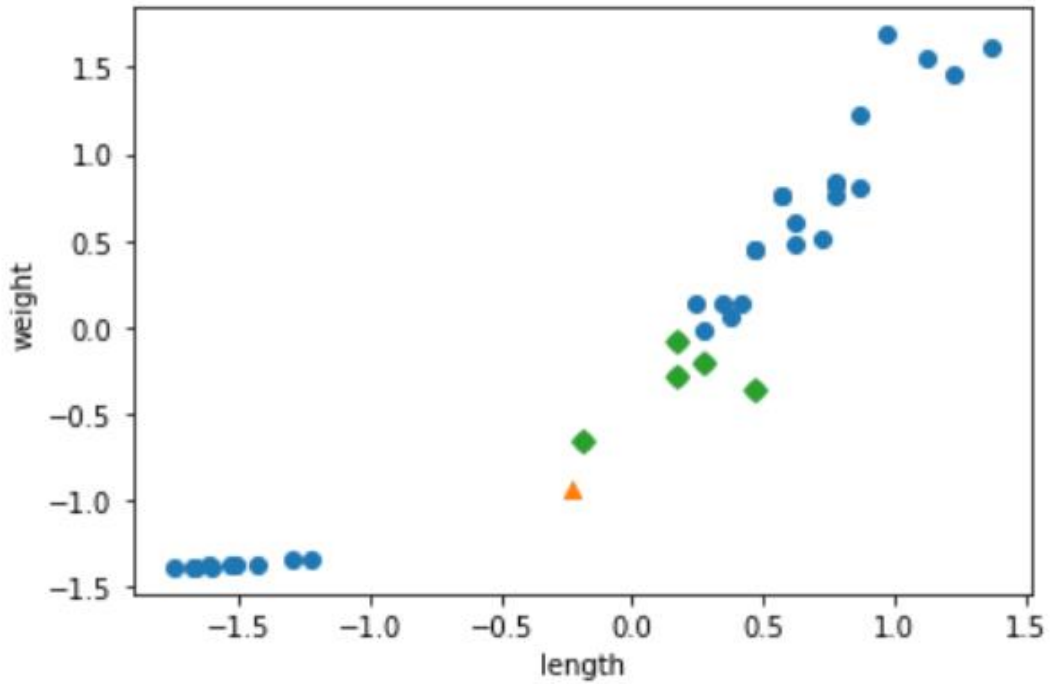
첫 번째 모델 : 빙어와 도미의 이진분류 – 오류 해결

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

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

```
[1.]
```

이제 1(도미)로 잘 예측하는 것을 확인할 수 있다.

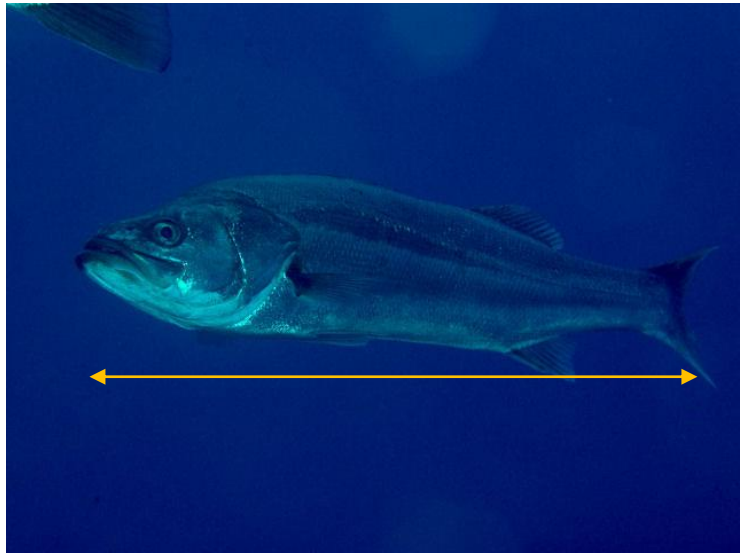


두 번째 모델 : 농어의 무게 예측(회귀)

```
import numpy as np

perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0, 21.0,
       21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5, 22.5, 22.7,
       23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3, 27.5, 27.5,
       27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0, 37.0, 37.0,
       39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0, 43.0, 43.5,
       44.0])
```

```
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
       1000.0])
```



1200g

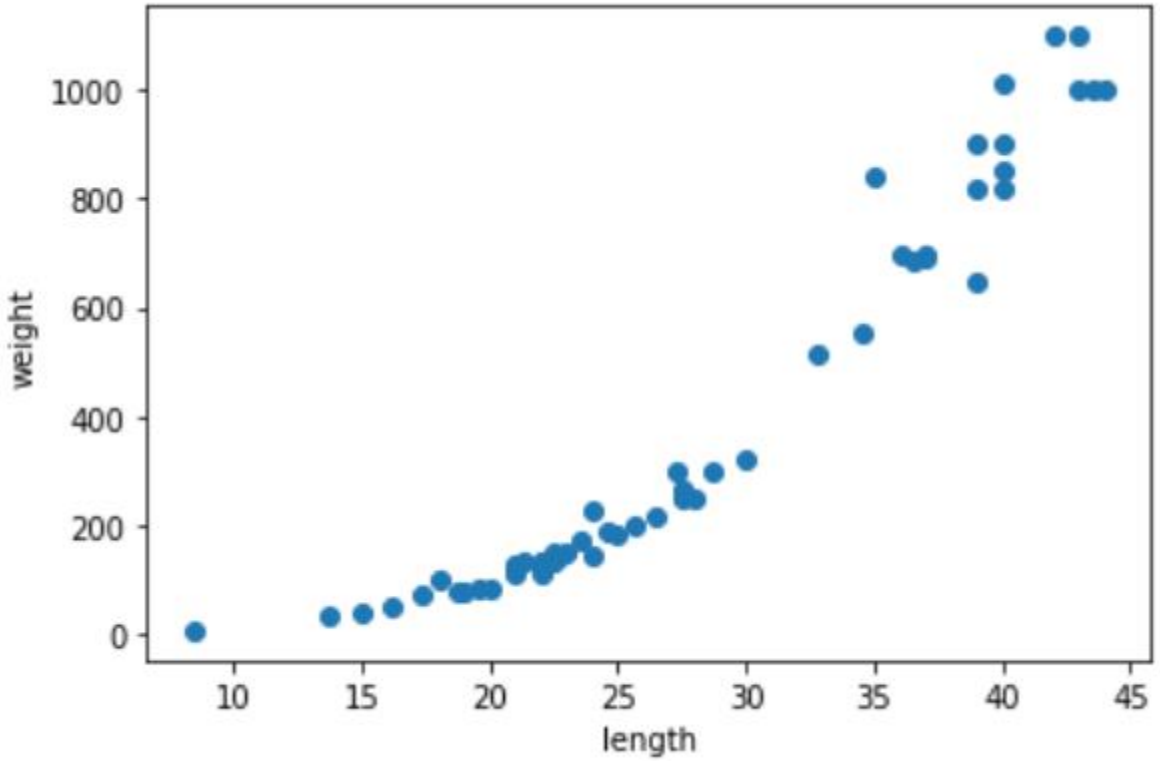
두 번째 모델 : 농어의 무게 예측(회귀) – 회귀(regression)

변수 간의 상관관계를 파악-> 변수가 주어졌을 때 특정 값 예측

Ex) 내년도 경제 성장률 예측, 배달 도착 시간 예측

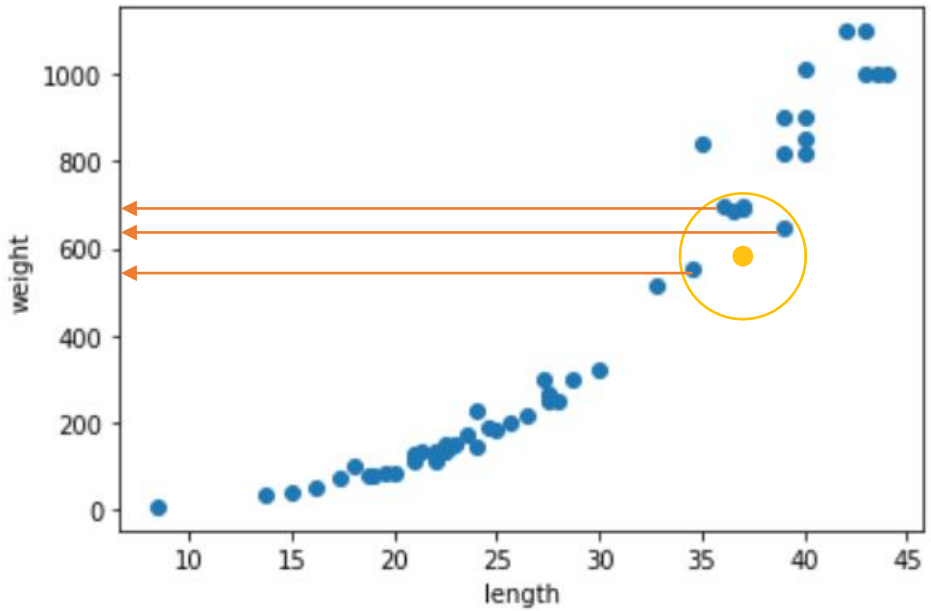
농어의 길이(cm) → 농어의 무게(g)

두 번째 모델 : 농어의 무게 예측(회귀) – 데이터 시각화



두 번째 모델 : 농어의 무게 예측(회귀) – k 최근접 이웃 회귀

이웃한 샘플의 평균으로 예측값 도출



두 번째 모델 : 농어의 무게 예측(회귀) – 훈련 및 평가

훈련 세트와 테스트 세트 나누기

```
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) # -1을 전달하면 나머지 원소 개수의 배열들로 모두 채우게 된다.
test_input = test_input.reshape(-1, 1)
print(train_input.shape, test_input.shape)
```

(42, 1) (14, 1) 행렬 형태로 변환

```
from sklearn.neighbors import KNeighborsRegressor

knr = KNeighborsRegressor()
```

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

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

0.992809406101064

분류에서는 정확하게 분류한 비율

회귀에선 정확하게 수치를 맞춘 비율? -> 불가능

두 번째 모델 : 농어의 무게 예측(회귀) – 결정계수(coefficient determination)

결정계수($\boldsymbol{R^2}$)

회귀에서 모델을 평가하는 값

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

예측이 정답(타깃)에 가까울수록 1에 근접 -> good

예측이 타깃 평균에 가까울 수록 0에 근접 -> bad

예측이 타깃 평균에 가까운 수준 => 성능 안 좋음

두 번째 모델 : 농어의 무게 예측(회귀) – 과대적합 과소적합

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

```
0.9698823289099254
```

훈련 세트의 점수가 테스트 세트의 점수에 비해 낮음

-> 과소적합

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

```
0.992809406101064
```

훈련 세트가 전체 데이터를 반영한다고 보기 때문에

훈련 세트의 점수가 **조금 더 높거나 비슷한 것이 이상적**

두 번째 모델 : 농어의 무게 예측(회귀) – 과대적합 과소적합

과대적합(overfitting)

: 훈련 세트에만 너무 잘 맞는 것.

- 모델이 과하게 복잡한 경우
- 필요 이상의 특징 파악

과소적합(underfitting)

: 훈련 세트에 적절히 훈련되지 않는 것.

- 모델이 너무 단순한 경우
- or 훈련 세트 테스트 세트의 크기가 작은 경우

->**모델을 좀 더 복잡하게 만들기**

두 번째 모델 : 농어의 무게 예측(회귀) – 과소적합 해결하기 (k 최근접 이웃 회귀)

K 최근접 이웃 회귀를 더 복잡하게

⇒ **더 적은 수의 이웃** 샘플 확인(기본값 5 -> 3)

⇒ 보다 좁은 범위에서 섬세하게 판단하겠다

```
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))
```

```
0.9746459963987609
```

두 번째 모델 : 농어의 무게 예측(회귀) – k 최근접 이웃 회귀의 한계

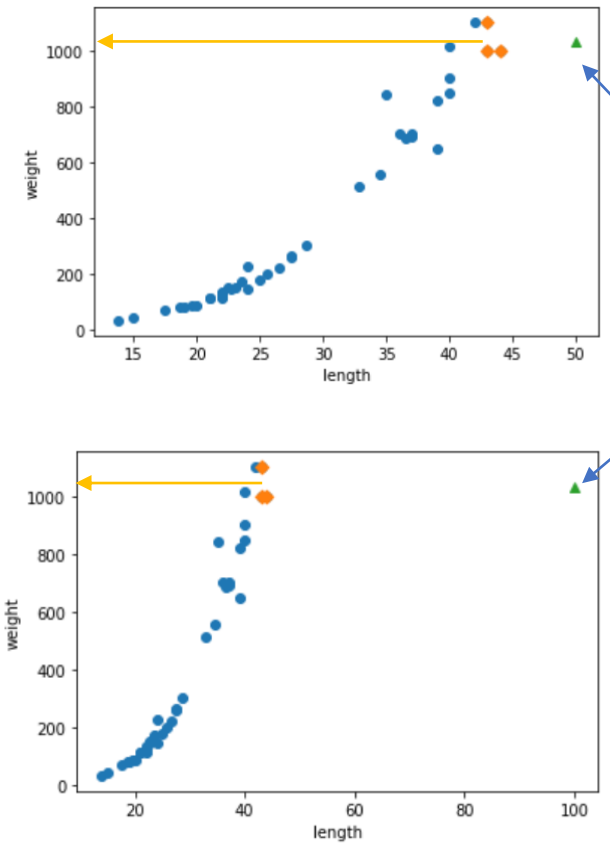
K 최근접 이웃 회귀 – 기존의 데이터 기반 예측 => **사례 기반 학습**

길이가 **50cm**인 농어 예측해보기

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

```
array([1033.33333333])
```

실제로는 길이 50cm 무게 1.5kg



기존 데이터에서 크게 벗어난
이례적인 데이터 예측 불가

How?

두 번째 모델 : 농어의 무게 예측(회귀) – 선형 회귀(linear regression)

데이터로 **직선의 방정식**을 구한다면
어떤 x(length)도 예측 가능

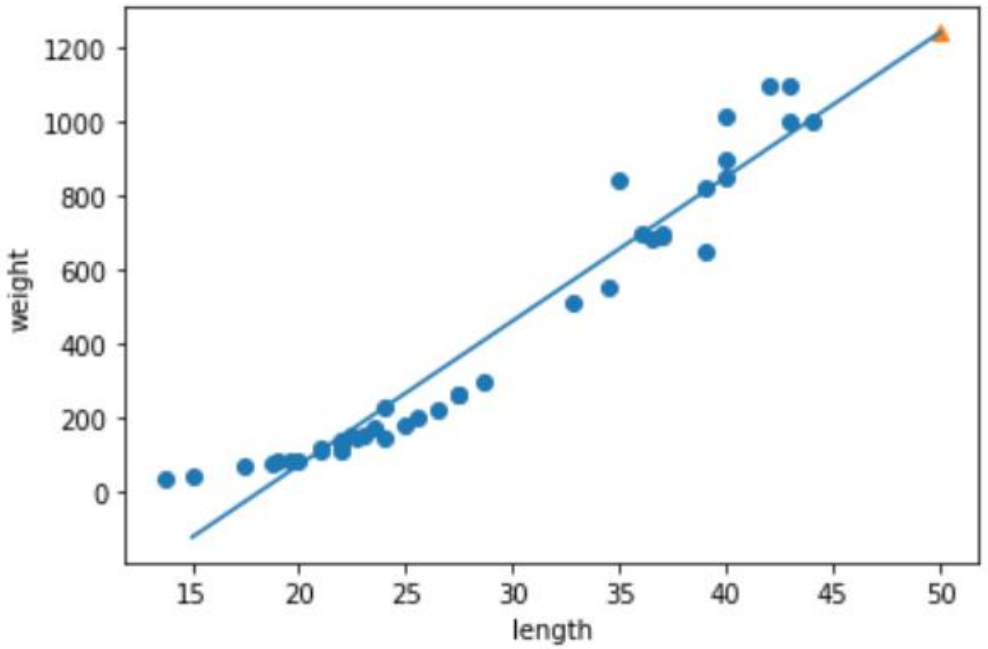
선형 회귀(Linear regression)

종속변수 Y와 독립변수 X와의 선형상관관계를 모델링

$$Y = aX + b$$

Y절편

계수(가중치)

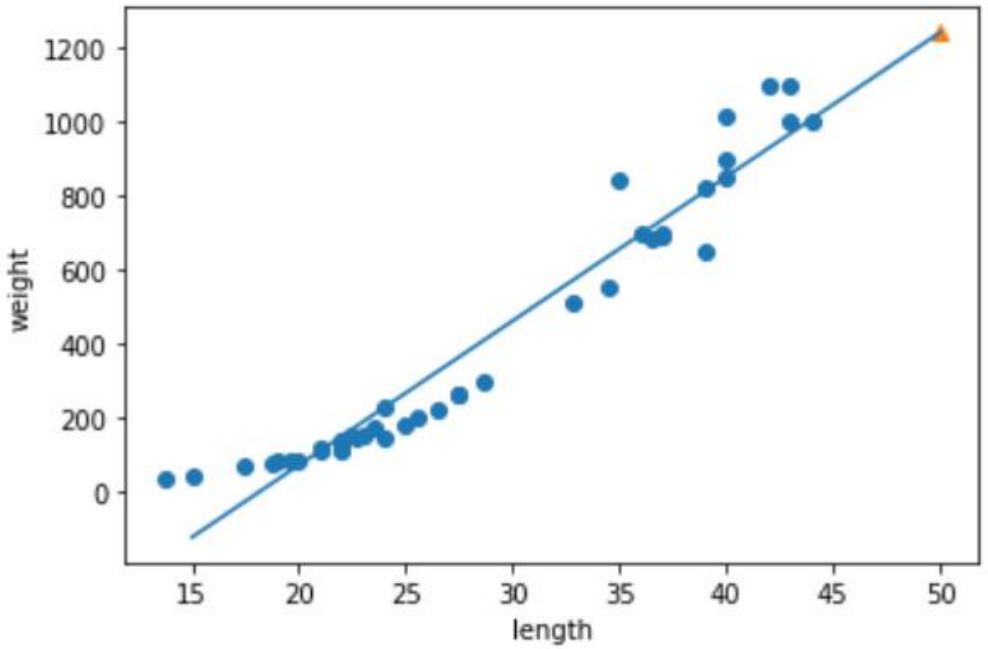


두 번째 모델 : 농어의 무게 예측(회귀) – 선형 회귀(linear regression)

데이터로 **직선의 방정식**을 구한다면
어떤 x(length)도 예측 가능

선형 회귀(Linear regression)

종속변수 Y와 독립변수 X와의 선형상관관계를 모델링



$Y = aX + b$

- Y절편
- 계수(가중치)
- 모델 파라미터

머신러닝 -> 최적의 모델 파라미터를 찾는 것

모델 기반 학습

두 번째 모델 : 농어의 무게 예측(회귀) – 선형 회귀 훈련

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

# 선형 회귀 모델 훈련
lr.fit(train_input, train_target)

# 50cm 농어에 대해 예측
print(lr.predict([[50]]))
```

```
[1241.83860323]
```

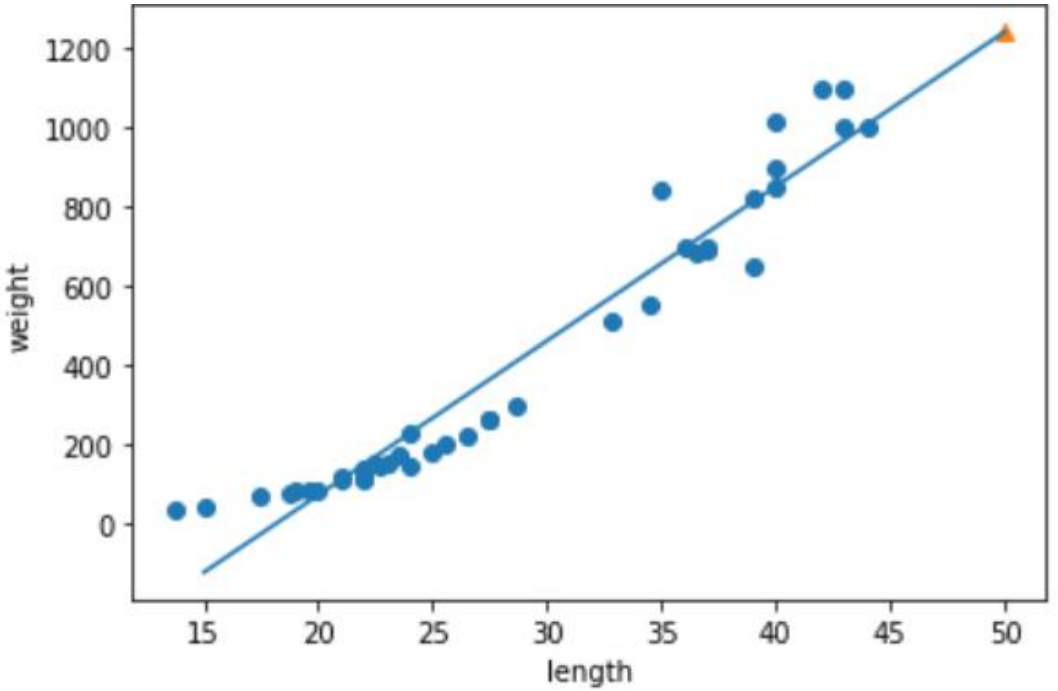
두 번째 모델 : 농어의 무게 예측(회귀) – 모델 파라미터 확인하기

```
print(lr.coef_, lr.intercept_)
```

```
[39.01714496] -709.0186449535477
```

가중치 Y절편

$$Weight = (39.01) * (Length) - 709.01$$



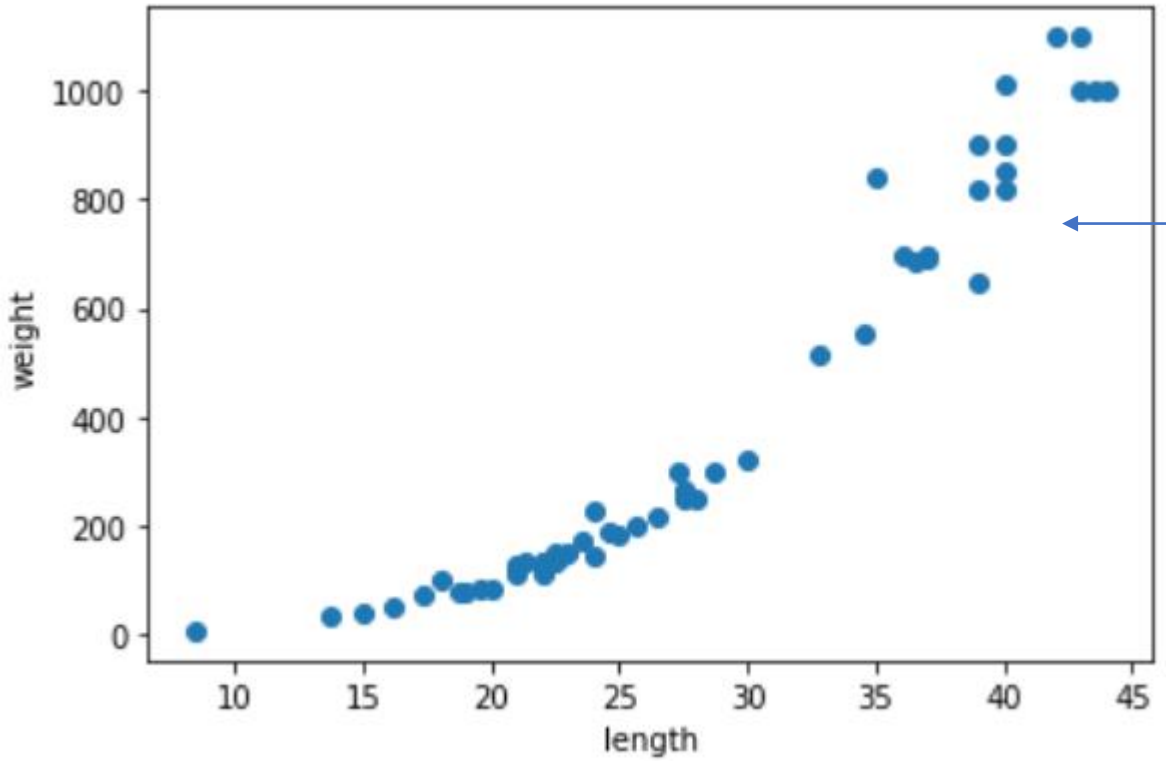
두 번째 모델 : 농어의 무게 예측(회귀) – 선형 회귀 평가

```
print(lr.score(train_input, train_target))
print(lr.score(test_input, test_target))
```

```
0.939846333997604
0.8247503123313558
```

-> 과소적합 => 모델이 너무 단순함

두 번째 모델 : 농어의 무게 예측(회귀) – 과소적합 원인



직선 보다는 곡선에 가깝다.

직선이 아닌 곡선을 활용.

-> 다항회귀

다항식을 활용한 선형 회귀

Ex) 2차 방정식

두 번째 모델 : 농어의 무게 예측(회귀) – 다항 회귀(polynomial regression) 훈련

```
train_poly = np.column_stack((train_input ** 2, train_input))
test_poly = np.column_stack((test_input ** 2, test_input))
```

x^2 데이터에 추가하기

```
lr = LinearRegression()
lr.fit(train_poly, train_target)

print(lr.predict([[50**2, 50]]))
```

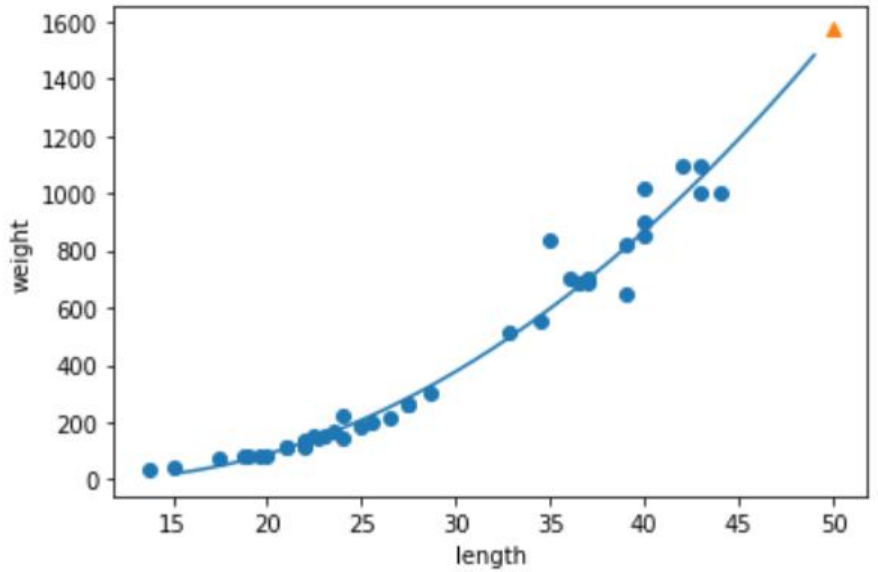
```
[1573.98423528]
```

실제 무게 1.5kg와 유사

두 번째 모델 : 농어의 무게 예측(회귀) – 다항 회귀 모델 파라미터 확인

```
print(lr.coef_, lr.intercept_)
```

```
[  1.01433211 -21.55792498] 116.0502107827827
```



$$(Weight) = (1.014) * (\text{Length})^2 - 21.557 * (Length) + 116.050$$

두 번째 모델 : 농어의 무게 예측(회귀) – 다항 회귀 평가

```
print(lr.score(train_poly, train_target))
print(lr.score(test_poly, test_target))
```

```
0.9706807451768623
0.9775935108325122
```

과소적합 -> 보다 복잡한 모델 필요

두 번째 모델 : 농어의 무게 예측(회귀) – 다중 회귀(Multiple Regression)

다중회귀(multiple regression)

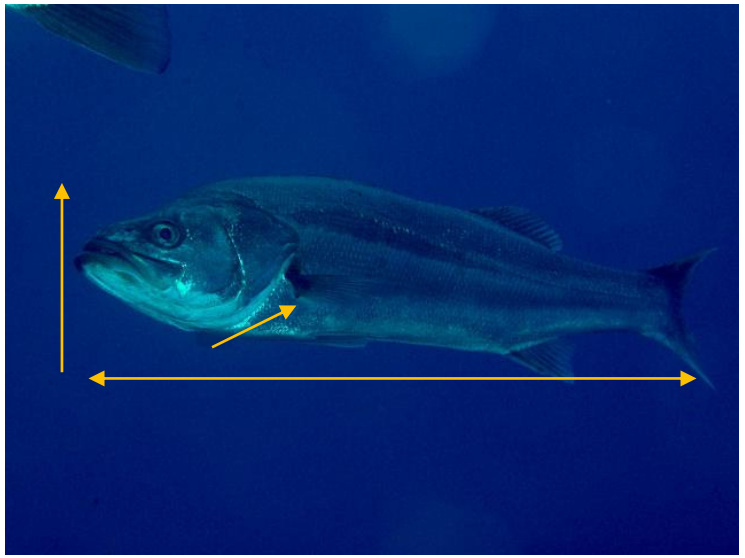
: 여러 개의 특성을 사용한 선형회귀

```
import pandas as pd
df = pd.read_csv('https://bit.ly/perch_csv_data')
perch_full = df.to_numpy() # 넘파이 배열로 변환
print(perch_full)
```

```
[[ 8.4   2.11  1.41]
 [13.7   3.53  2.  ]
 [15.    3.82  2.43]
 [16.2   4.59  2.63]
 [17.4   4.59  2.94]
 [18.    5.22  3.32]
 [18.7   5.2   3.12]
```

길이 높이 너비

```
import numpy as np
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
       1000.0])
```



1200g

두 번째 모델 : 농어의 무게 예측(회귀) – 특성 공학(feature engineering)

특성공학

: 기존의 특성으로 새로운 특성을 만들어내는 것.

길이, 높이, 너비, 길이x높이, 길이x너비, 길이x너비, …

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly = PolynomialFeatures(include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
print(train_poly.shape)
```

```
(42, 9)
```

3개의 특성을 9개로 만듦.

두 번째 모델 : 농어의 무게 예측(회귀) – 다중 회귀 훈련 및 평가

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(train_poly, train_target)
print(lr.score(train_poly, train_target))
```

0.9903183436982124

```
print(lr.score(test_poly, test_target))
```

0.9714559911594134

⇒ 과소 적합 해결

두 번째 모델 : 농어의 무게 예측(회귀) – 특성의 수가 많을수록 좋은가?

```
poly = PolynomialFeatures(degree=5, include_bias=False) #degree로 최대 차수를 지정.
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)
print(train_poly.shape)
```

(42, 55)

특성을 55개까지 늘림

```
lr.fit(train_poly, train_target)
print(lr.score(train_poly, train_target))
```

0.9999999999991097

```
print(lr.score(test_poly, test_target))
```

-144.40579242684848

과대적합

-> 필요이상으로 모델이 복잡함

두 번째 모델 : 농어의 무게 예측(회귀) – 규제(regularization)

규제(regularization)

: 과도하게 학습하지 못하도록 훼방하는 것

How? -> 가중치의 크기를 작게

- 릿지(ridge) 회귀
- 라쏘(lasso) 회귀 (계수를 아예 0으로 만들 수도 있음)

두 번째 모델 : 농어의 무게 예측(회귀) – 릿지 회귀

계수를 제곱한 값을 기준으로 규제 적용

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_poly) # 훈련 세트로 학습한 변환기
train_scaled = ss.transform(train_poly)
test_scaled = ss.transform(test_poly) # 테스트 세트는 꼭 훈련 세트로 학습한 변환기를 사용해 변환해야 한다.
```

특성 스케일 정규화

```
from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(train_scaled, train_target)
print(ridge.score(train_scaled, train_target))
```

```
0.98961016710373 43
```

```
print(ridge.score(test_scaled, test_target))
```

```
0.9790693977615397
```

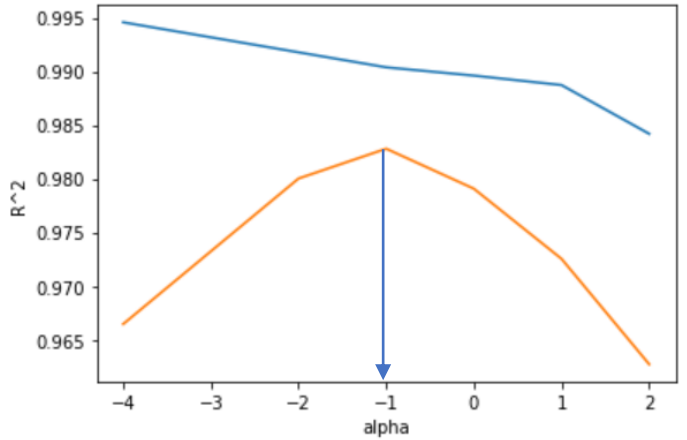
두 번째 모델 : 농어의 무게 예측(회귀) – 규제의 강도 조절 (alpha값 조절)

Alpha 값이 클수록 규제의 강도 세짐. (하이퍼 파라미터)

적절한 Alpha값? -> 훈련 세트 점수와 테스트 세트 점수가 가장 가까운 지점 (**가장 적은 괴리율**)

```
alpha_list = [0.0001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
  ridge = Ridge(alpha=alpha)
  ridge.fit(train_scaled, train_target)
  train_score.append(ridge.score(train_scaled, train_target))
  test_score.append(ridge.score(test_scaled, test_target))
```

```
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.show()
```



최적 alpha = 0.1

두 번째 모델 : 농어의 무게 예측(회귀) – 릿지 회귀 with 최적 alpha

```
ridge =Ridge(alpha=0.1)
ridge.fit(train_scaled, train_target)
print(ridge.score(train_scaled, train_target))
print(ridge.score(test_scaled, test_target))
```

```
0.9903815817570366
0.9827976465386926
```

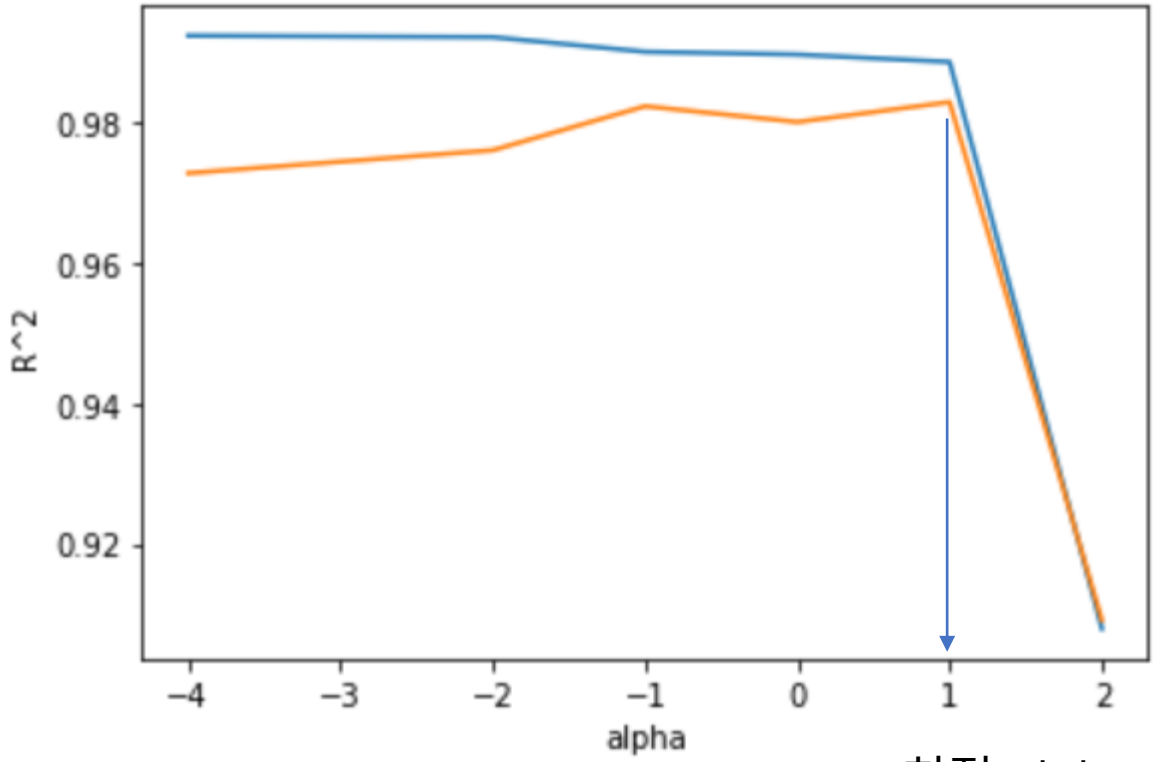
두 번째 모델 : 농어의 무게 예측(회귀) – 라쏘 회귀

```
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(train_scaled, train_target)
print(lasso.score(train_scaled, train_target))
```

0.9897898972080961

```
print(lasso.score(test_scaled, test_target))
```

0.9800593698421883



최적 alpha = 10

두 번째 모델 : 농어의 무게 예측(회귀) – 라쏘 회귀 with 최적 alpha

```
lasso = Lasso(alpha=10)
lasso.fit(train_scaled, train_target)
print(lasso.score(train_scaled, train_target))
print(lasso.score(test_scaled, test_target))
```

```
0.9888067471131867
0.9824470598706695
```

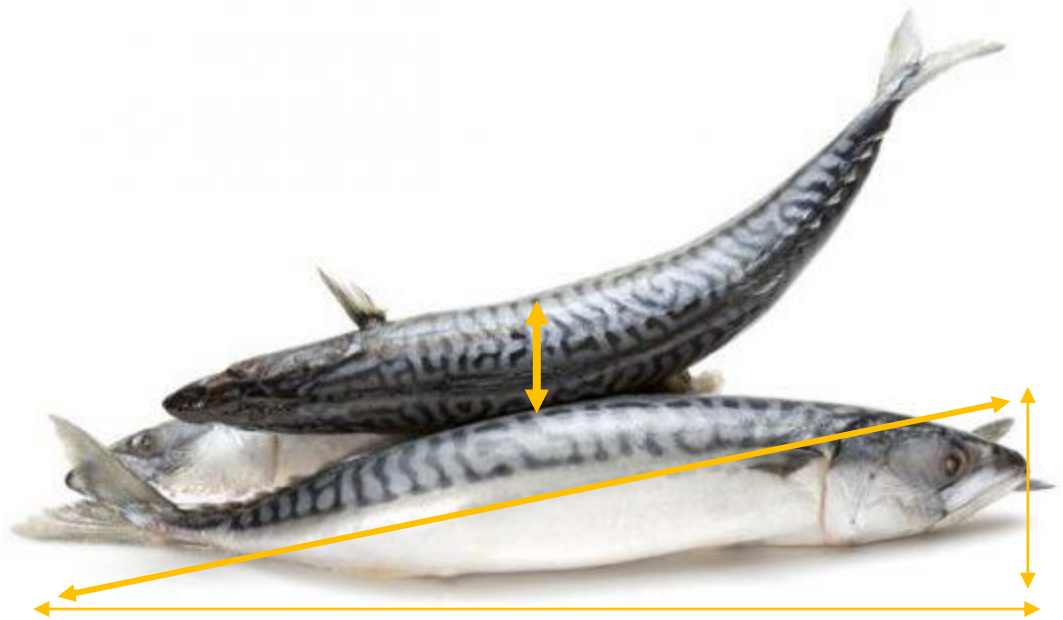
```
print(np.sum(lasso.coef_ == 0))
```

```
40
```

55개의 특성 중 15개만 사용

-> 유용한(의미 있는) 특성을 골라내는 데 사용가능

세 번째 모델 : 생선 종 예측(로지스틱 회귀)



500g

	Weight	Length	Diagonal	Height	Width

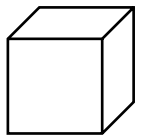
['Bream' 'Roach' 'Whitefish' 'Parkki' 'Perch' 'Pike' 'Smelt']

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 로지스틱 회귀(Logistic Regression)

이름은 회귀이지만 다중 클래스 분류 모델. 선형 방정식을 구해 z 값 산출.

$$z = a(Weight) + b(Length) + c(Diagonal) + d(Height) + e(Width) + f$$

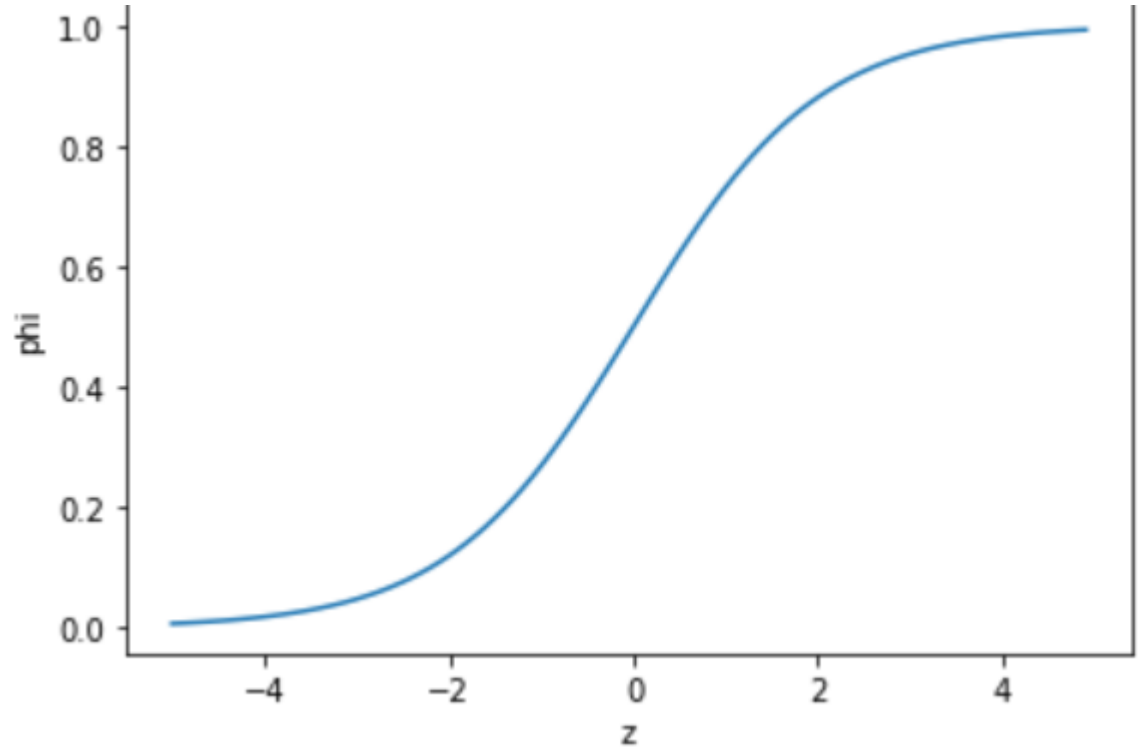
Z는 실수, 어떻게 확률로 변환?



세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 시그모이드 함수(sigmoid function)

실수를 0과 1 사이의 값으로 변환 -> 확률

$$\frac{1}{(1+e^{-z})}$$



세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 이진 분류

```
# 로지스틱 회귀 모델 훈련해보기

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_bream_smelt, target_bream_smelt)
```

```
# 처음 5개 샘플 예측해보기

print(lr.predict(train_bream_smelt[:5]))
```

```
['Bream' 'Smelt' 'Bream' 'Bream' 'Bream']
```

```
# 판단 확률보기

print(lr.predict_proba(train_bream_smelt[:5]))
```

```
[[0.99759855 0.00240145]
 [0.02735183 0.97264817]
 [0.99486072 0.00513928]
 [0.98584202 0.01415798]
 [0.99767269 0.00232731]]
```

도미 빙어

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 이진 분류 선형방정식 구해보기

```
# 로지스틱 회귀가 학습한 계수(가중치) 확인해 보기

print(lr.coef_, lr.intercept_)
```

```
[[-0.4037798  -0.57620209 -0.66280298 -1.01290277 -0.73168947]] [-2.16155132]
```

즉, 다음의 방정식을 구한 것이다.

z = -0.404 x (weight) -0.576 x (Length) -0.663 x (Diagonal) -1.013 x (Height) -0.732 x (Width) -2.161

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – z 값 시그모이드 함수 적용

```
# z값 실제로 구해보기

decisions = lr.decision_function(train_bream_smelt[:5])
print(decisions)
```

```
[-6.02927744  3.57123907 -5.26568906 -4.24321775 -6.0607117 ]
```

```
# 시그모이드 함수 적용해 실제 확률값 구하기

from scipy.special import expit
print(expit(decisions))
```

```
[0.00240145 0.97264817 0.00513928 0.01415798 0.00232731]
```

```
[[0.99759855 0.00240145]
 [0.02735183 0.97264817]
 [0.99486072 0.00513928]
 [0.98584202 0.01415798]
 [0.99767269 0.00232731]]
```

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 다중 분류

```
# 로지스틱 회귀 다중 분류 모델 훈련하기

lr = LogisticRegression(C=20, max_iter=1000)
lr.fit(train_scaled, train_target)
print(lr.score(train_scaled, train_target))
print(lr.score(test_scaled, test_target))
```

```
0.9327731092436975
0.925
```

```
# 처음 5개 샘플 예측 보기

print(lr.predict(test_scaled[:5]))
```

```
['Perch' 'Smelt' 'Pike' 'Roach' 'Perch']
```

```
# 예측 확률보기

proba = lr.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=3)) # 소수점 네 번째 자리에서 반올림
```

```
[[0.     0.014 0.841 0.     0.136 0.007 0.003]
 [0.     0.003 0.044 0.     0.007 0.946 0.    ]
 [0.     0.    0.034 0.935 0.015 0.016 0.    ]
 [0.011 0.034 0.306 0.007 0.567 0.    0.076]
 [0.     0.    0.904 0.002 0.089 0.002 0.001]]

['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']
```

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 다중 분류 모델 파라미터 확인

```
print(lr.coef_.shape, lr.intercept_.shape)
```

```
(7, 5) (7,)
```

5개의 계수가 총 7세트 -> **클래스 수 만큼** 선형방정식 존재

= **클래스 별로 z값 산출**

어떻게 확률로?

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 소프트 맥스(softmax)

$$e\_sum = e^{z1} + e^{z2} + e^{z3} + e^{z4} + e^{z5} + e^{z6} + e^{z7}$$

$$s1 = \frac{e^{z1}}{e\_sum},\ s2 = \frac{e^{z2}}{e\_sum},\ \cdots,\ s7 = \frac{e^{z7}}{e\_sum}$$

모두 더하면 1

세 번째 모델 : 생선 종 예측(로지스틱 회귀) – 소프트 맥스(softmax)

```
# 처음 5개 샘플의 z1~z7 구하기

decision = lr.decision_function(test_scaled[:5])
print(np.round(decision, decimals=2))
```

```
[[ -6.5    1.03   5.16  -2.73   3.34   0.33  -0.63]
 [-10.86   1.93   4.77  -2.4    2.98   7.84  -4.26]
 [ -4.34  -6.23   3.17   6.49   2.36   2.42  -3.87]
 [ -0.68   0.45   2.65  -1.19   3.26  -5.75   1.26]
 [ -6.4   -1.99   5.82  -0.11   3.5   -0.11  -0.71]]
```

```
# 소프트 맥스로 확률 구하기

from scipy.special import softmax
proba = softmax(decision, axis=1) # 각 행, 즉 각 샘플에 대
print(np.round(proba, decimals=3))
```

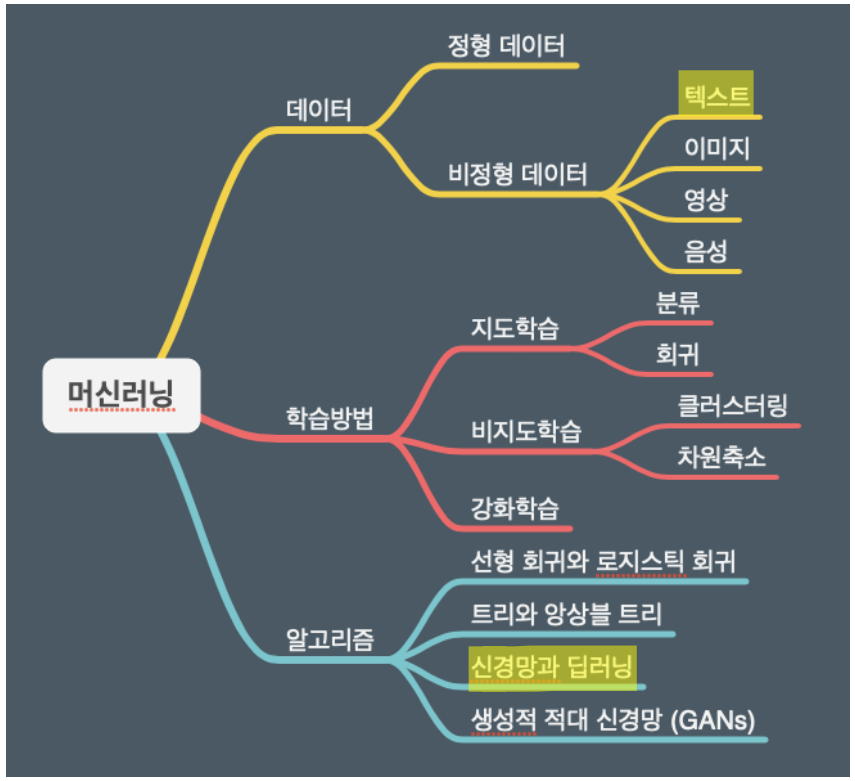
```
[[0.     0.014 0.841 0.     0.136 0.007 0.003]
 [0.     0.003 0.044 0.     0.007 0.946 0.    ]
 [0.     0.    0.034 0.935 0.015 0.016 0.    ]
 [0.011 0.034 0.306 0.007 0.567 0.    0.076]
 [0.     0.    0.904 0.002 0.089 0.002 0.001]]
```

```
# 예측 확률보기

proba = lr.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=3)) # 소수점 네 번째 자리에서 반올림
```

```
[[0.     0.014 0.841 0.     0.136 0.007 0.003]
 [0.     0.003 0.044 0.     0.007 0.946 0.    ]
 [0.     0.    0.034 0.935 0.015 0.016 0.    ]
 [0.011 0.034 0.306 0.007 0.567 0.    0.076]
 [0.     0.    0.904 0.002 0.089 0.002 0.001]]
```

What's Next?


머신러닝
데이터
정형 데이터
비정형 데이터
텍스트
이미지
영상
음성
학습방법
지도학습
분류
회귀
비지도학습
클러스터링
차원축소
강화학습
알고리즘
선형 회귀와 로지스틱 회귀
트리와 앙상블 트리
신경망과 딥러닝
생성적 적대 신경망 (GANs)


O'REILLY
Natural Language Processing with
PyTorch
파이토치로 배우는 자연어 처리
딥러닝을 이용한
자연어 처리
애플리케이션 구축
한빛미디어
델립 라오, 브라이언 맥머핸 지음
박해선 옮김