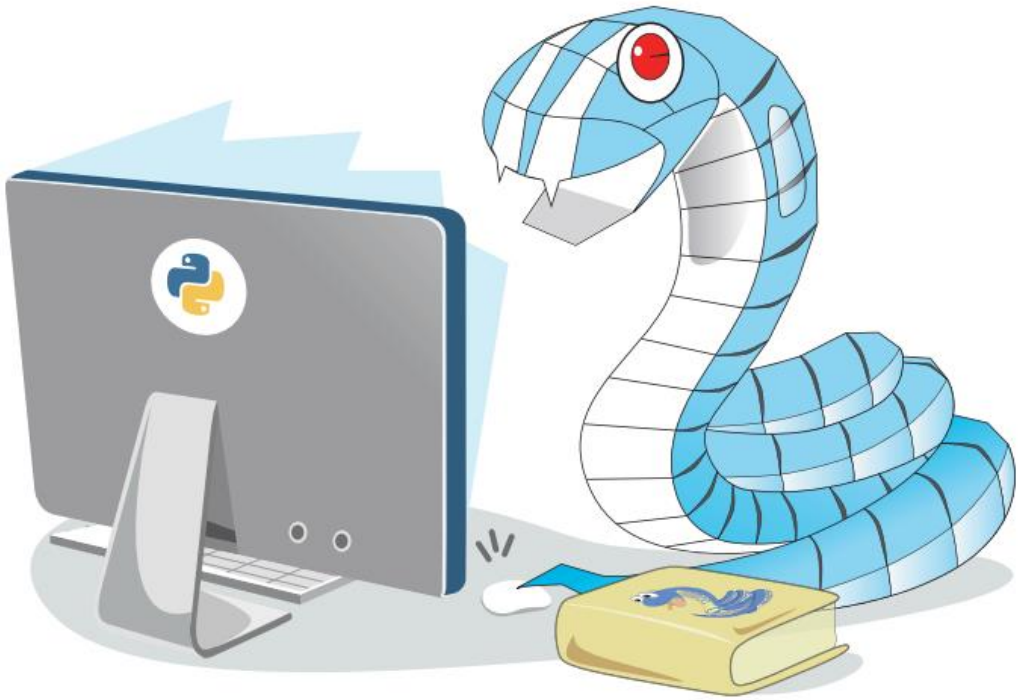


머신러닝

로지스틱회귀

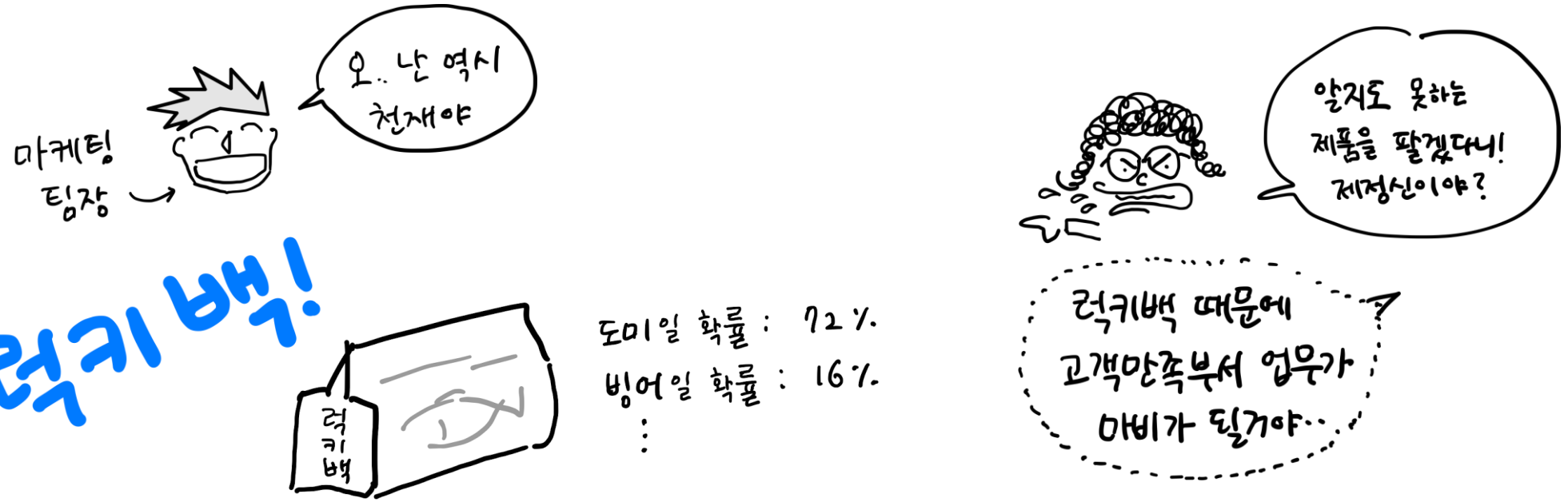
회귀분석 종류

- 선형 회귀 Linear Regression 모델: 수치 예측
  - 연속변수에 대한 분석 모형
  - 단순 선형 회귀: 독립변수 1개
  - 다중 선형 회귀: 독립변수 2개 이상
  - 다항 회귀(Polynomial Regression)
    - 독립변수를 다항식으로 변환하여 선형 회귀 수행

- 로지스틱 회귀 모델: 범주 예측
  - 범주형 변수에 대한 분석 모형
  - 이항 로지스틱 회귀: 이항 분류
  - 다항 로지스틱 회귀: 다항 분류

새로운 도전: 럭키 백 이벤트 기획

- 럭키 백은 구성품을 모른 채 고객은 제품 구매 -> 제품 개봉 후 구성품 알 수 있다.
  - 고객이 선호하지 않은 제품의 경우에는?
  - 럭키 백에 포함된 생선(제품)의 확률을 미리 알려주므로 고객의 의사결정에 도움을 준다.
    - **포함된 생선 확률을 구하는 문제**

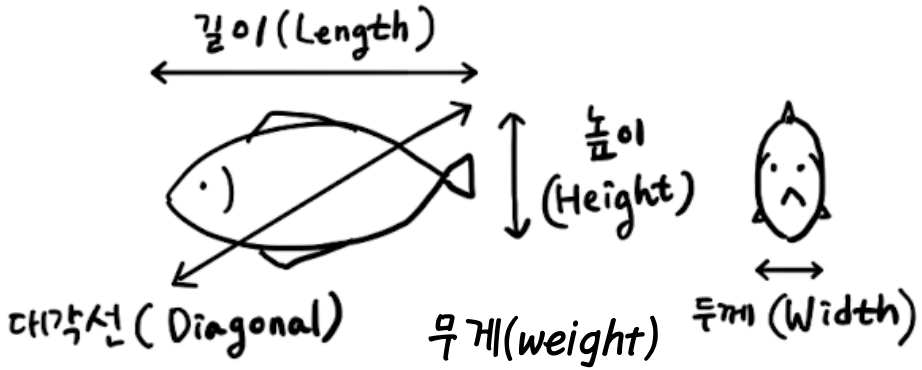


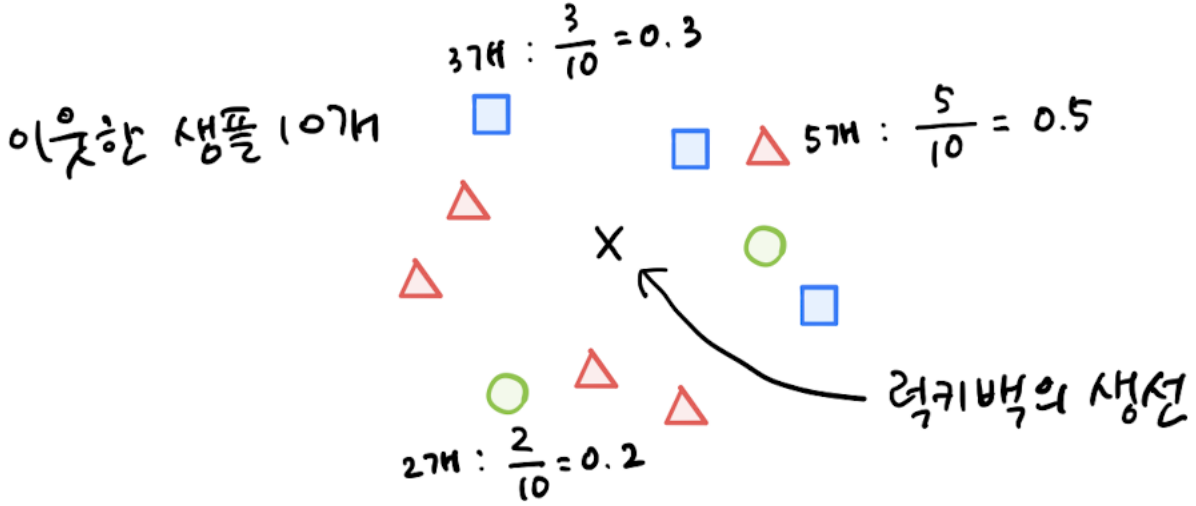
• 새로운 도전: 럭키 백 이벤트 기획

[럭키백에 들어갈 수 있는 생선 7가지]

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

K-NN 알고리즘 적용이 가능한가?
최근접 이웃을 찾아서 그 샘플이 속한 클래스 비율을 확률로 출력하면 어떨까?





K-NN 분류 모델: KNeighborsClassifier 클래스

1. 훈련 데이터 준비

```
import pandas as pd

fish = pd.read_csv('https://bit.ly/fish_csv_data')
fish.head()
```

	Species	Weight	Length	Diagonal	Height	Width
0	Bream	242.0	25.4	30.0	11.5200	4.0200
1	Bream	290.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	26.5	31.1	12.3778	4.6961
3	Bream	363.0	29.0	33.5	12.7300	4.4555
4	Bream	430.0	29.0	34.0	12.4440	5.1340

```
#데이터셋으로 변환하기
fish_input = fish[['Weight','Length','Diagonal','Height','Width']].to_numpy()

fish_target = fish['Species'].to_numpy()
```

K-NN 분류 모델 : KNeighborsClassifier 클래스

2. 생선의 종류(클래스) 확인

```
#pd.unique() 특정열에서 고유한 값 추출하기 위한 함수
print(pd.unique(fish['Species']))
```

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

```
#판단스로 저장된 데이터와 넘파이로 변환된 데이터 확인하기
print(fish.head())   #판단스 데이터 중 앞부분 5개
print(fish_input[:5])
print(fish_target[:5])
```

K-NN 분류 모델 : KNeighborsClassifier 클래스

3. 데이터셋 분할과 표준화

```
from sklearn.model_selection import train_test_split

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


from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_input)  #훈련데이터의 기준 측정
train_scaled = ss.transform(train_input)        #측정된 기준으로 훈련 데이터 표준화
test_scaled = ss.transform(test_input)          #측정된 기준으로 테스트 데이터 표준화

#변환된 데이터 5개씩 출력 확인
```

K-NN 분류 모델 : KNeighborsClassifier 클래스

4. KNeighberClassifier 클래스로 KNN 모델 생성, 평가, 예측

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(train_scaled, train_target)

print(kn.score(train_scaled, train_target))
print(kn.score(test_scaled, test_target))
```

다음 데이터는 어떤 클래스에 속할까?

	Weight	Length	Diagonal	Height	Width
Data 1	[290.,	26.3,	31.2,	12.48,	4.3056]
Data 2	[340.,	26.5,	31.1,	12.3778,	4.6961]
Data 3	[363.,	29.,	33.5,	12.73,	4.4555]

K-NN 분류 모델 : KNeighborsClassifier 클래스

4. KNeighberClassifier 클래스로 KNN 모델 생성, 평가, 예측

```
#새로운 샘플 데이터 클래스 예측하기
#step1. 새로운 데이터를 넘파이 배열로 변환
new_sample=np.array([[290.,26.3,31.2,12.48,4.3056],[340.,26.5,31.1,12.3778,4.6961],[363.,29.,33.5,
12.73,4.4555]])

#step2. 새로운 데이터 표준화하기(기준은 훈련데이터)
new_sample_scaled = ss.transform(new_sample)

#step3. 새로운 데이터 클래스 예측하기
kn.predict(new_sample_scaled)
```

K-NN 분류 모델 : KNeighborsClassifier 클래스

5. KNN 분류 모델의 예측 확률 계산

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(train_scaled, train_target)

print(kn.classes_)
['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']
```

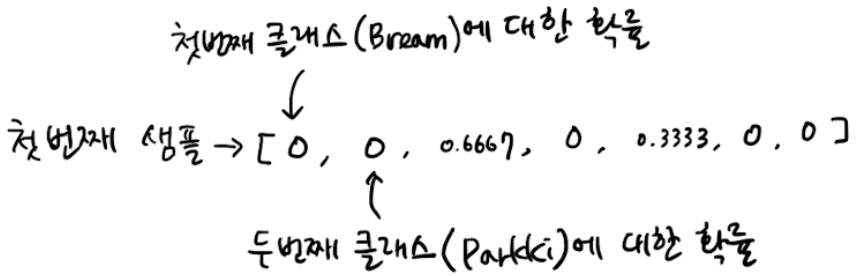
← 데이터 분류 클래스

```
print(kn.predict(test_scaled[:5]))
['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']
```

← 테스트 데이터 5개의 클래스 예측

```
proba = kn.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=4))
[[0.     0.     1.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     1.     0.    ]
 [0.     0.     0.     1.     0.     0.     0.    ]
 [0.     0.     0.6667 0.     0.3333 0.     0.    ]
 [0.     0.     0.6667 0.     0.3333 0.     0.    ]]
```

← 테스트 데이터 5개의 클래스 예측 확률



K-NN 분류 모델 : KNeighborsClassifier 클래스

6. 테스트셋의 4번째 데이터의 최근접 이웃 클래스 추출

```
#테스트 데이터 중 4번째 데이터의 최근접 이웃 클래스 3개 선택하기
distances, indexes = kn.kneighbors(test_scaled[3:4])
#최근접 이웃 클래스 출력
print(train_target[indexes])
```

```
[['Roach' 'Perch' 'Perch']]
```

→ Perch 클래스로 예측함

***KNN의 k값을 3을 정했으므로 클래스 예측 확률은 0, 1/3, 2/3, 1 중 하나임.

분류 모델: LogisticRegression 클래스

➢ 로지스틱 회귀 알고리즘(확률 예측을 통한 분류 기법)

- 로지스틱 회귀(Logistic Regression)는 회귀를 사용하여 데이터가 어떤 범주에 속할 확률을 0에서 1 사이의 값으로 예측하고 그 확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해주는 지도 학습 알고리즘이다.
- 로지스틱 회귀 알고리즘의 결과 값은 '**분류를 위한 확률**'이고, 그래서 이 확률이 특정 수준 이상 확보되면 샘플이 그 클래스에 속할지 말지 결정할 수 있다.
- 시그모이드, 소프트맥스 함수를 사용하여 클래스 확률을 출력한다.
  - 시그모이드 함수: 로지스틱 함수라고도 함. 선형방정식의 출력을 0과 1 사이의 값으로 압축하며 이진 분류를 위해 사용됨
  - 소프트맥스 함수: 여러개의 선형 방정식의 출력값을 0~ 1사이로 정규화하여 모든 클래스의 확률 합이 1이 되도록 만들어 주는 함수.

이항 분류: LogisticRegression 클래스

➢ 로지스틱 회귀 알고리즘

- 선형회귀식

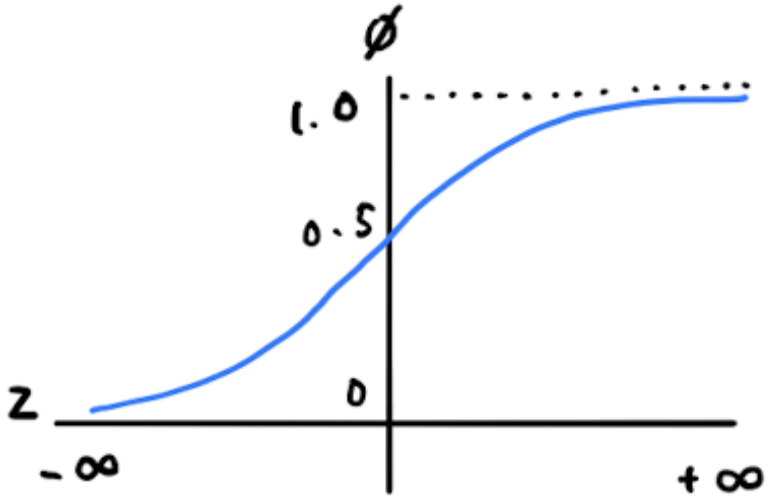
$$z = a \times 무게 + b \times 길이 + c \times 대각선 + d \times 높이 + e \times 두께 + f$$

으로 선형회귀모델 생성-> z 계산 -> z를 시그모이드 함수에 입력 (ø(z)) -> 결과 출력

- 결과는 0과 1 사이의 값을 가짐
  - z 값이 커질수록 출력은 1에 가까워지고, z 값이 작아질수록 출력은 0에 가까워진다.

- 시그모이드 함수(sigmoid function) 정의

$$\phi = \frac{1}{1 + e^{-z}}$$

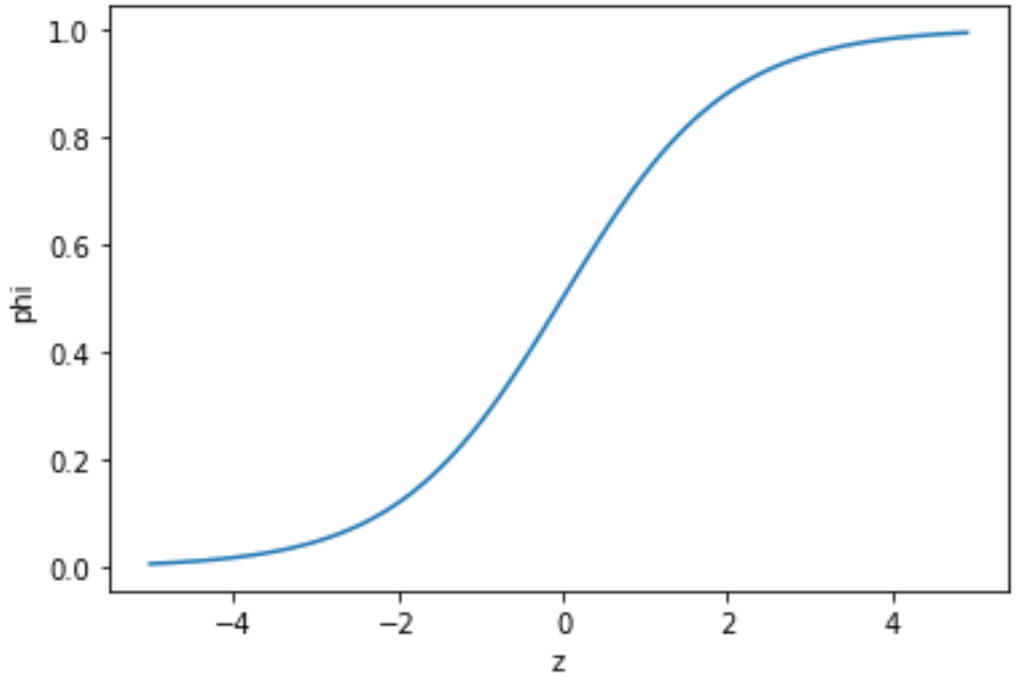


시그모이드 그래프

이항 분류: LogisticRegression 클래스

- 시그모이드 그래프 그리기

```
import numpy as np
import matplotlib.pyplot as plt
z = np.arange(-5, 5, 0.1)
phi = 1 / (1+np.exp(-z))
plt.plot(z, phi)
plt.xlabel('z')
plt.ylabel('phi')
plt.show()
```



이항 분류: LogisticRegression 클래스

```
bream_smelt_indexes = (train_target == 'Bream') | (train_target == 'Smelt')
train_bream_smelt = train_scaled[bream_smelt_indexes]
target_bream_smelt = train_target[bream_smelt_indexes]
```

훈련세트에서 도미(Bream)와
빙어(Smelt) 행 추출
-> 2개 클래스에 대한 데이터셋 구성

```
from sklearn.linear_model import LogisticRegression
```

```
lr = LogisticRegression()
lr.fit(train_bream_smelt, target_bream_smelt)
```

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

```
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]]
```

Bream일 확률 Smelt일 확률

이항 분류: LogisticRegression 클래스

➢ 로지스틱 회귀 계수 확인

```
print(lr.coef_, lr.intercept_)
[[-0.4037798  -0.57620209 -0.66280298 -1.01290277 -0.73168947]] [-2.16155132]
```

$$z = -0.404 \times \text{무게} - 0.576 \times \text{길이} - 0.663 \times \text{대각선} - 0.013 \times \text{높이} - 0.732 \times \text{두께} - 2.161$$

```
decisions = lr.decision_function(train_bream_smelt[:5])
print(decisions)
[-6.02927744  3.57123907 -5.26568906 -4.24321775 -6.0607117 ]
```

← z 계산

```
from scipy.special import expit
print(expit(decisions))
[0.00240145 0.97264817 0.00513928 0.01415798 0.00232731]
```

← 시그모이드 함수 모듈

← 시그모이드 함수 출력값은 Predict_prob() 출력의 2열(양성클래스 예측값)과 동일

$$\phi = \frac{1}{1 + e^{-z}}$$

다중 분류 문제

LogisticRegression, Softmax

다중 분류: LogisticRegression 클래스

➢ 다중 분류 문제: 소프트맥스함수로 해결!!

Softmax 함수는 각 클래스에 속할 확률을 계산하여 가장 높은 확률을 가진 클래스를 선택한다.

1. 각 클래스 k에 대해 선형 결합을 계산

$$z_k = w_{k1}x_{k1} + w_{k2}x_{k2} + w_{kn}x_{kn} + b_k$$

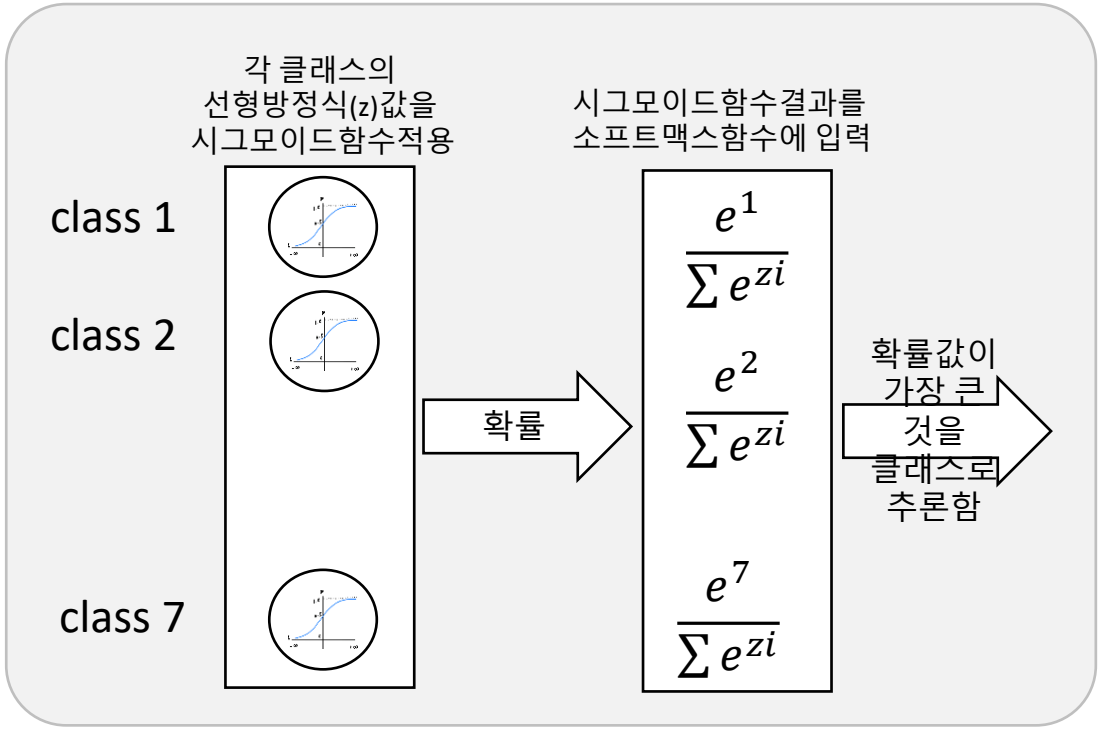
2. 시그모이드 함수로 각 클래스로 분류될 확률 계산

$$\phi = \frac{1}{1 + e^{-z}}$$

2. 소프트맥스 함수 적용
각 클래스의 확률을 계산

$$P(y = k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

3. 확률이 가장 높은 클래스를 선택



다중 분류: LogisticRegression 클래스

```
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
```

- c 규제 제어 매개변수(기본값 1)- 작을수록 규제가 커짐(LogisticRegression은 기본적으로 L2 규제 적용함)
- max_iter 매개변수를 이용해 반복 횟수를 지정, 기본값은 100(반복 횟수가 부족하면 경고 발생)

다중 분류: LogisticRegression 클래스

```
proba = lr.predict_proba(test_scaled[:5])
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]]
```

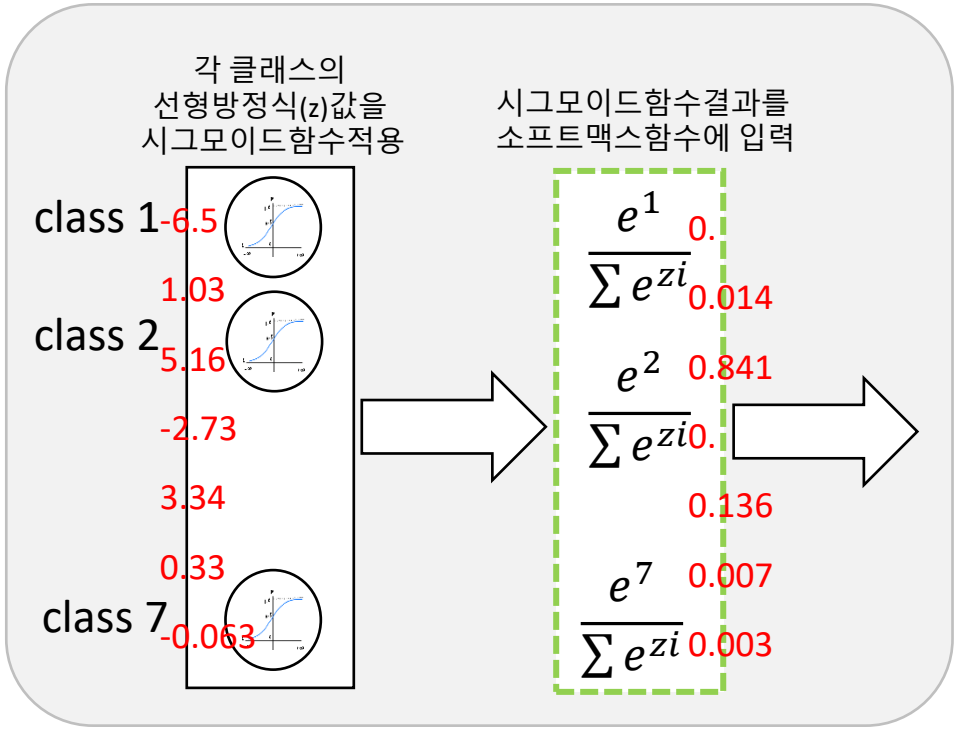
```
print(lr.coef_.shape, lr.intercept_.shape)
(7, 5) (7,)
```

다중 분류: LogisticRegression 클래스

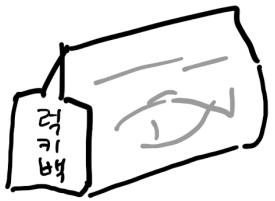
- LogisticRegression 이용한 다중 분류 결과와 소프트맥스 함수 결과 비교

```
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]]
```

```
from scipy.special import softmax

proba = softmax(decision, axis=1)
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]]
```



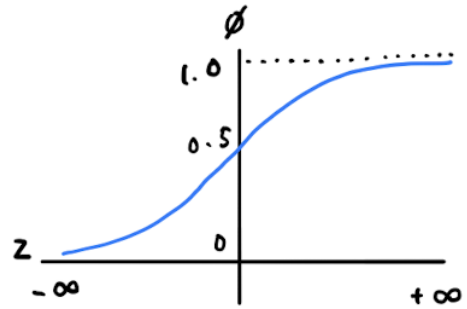
Review

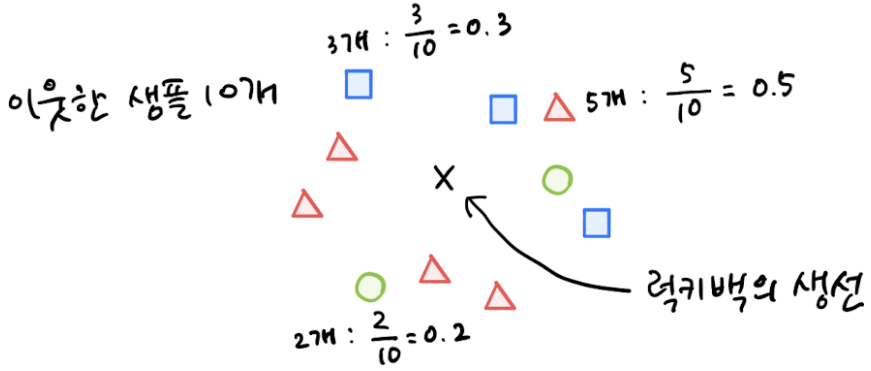


도미일 확률 : 72%
빙어일 확률 : 16%
⋮

$$z = a \times \text{무게} + b \times \text{길이} + c \times \text{대각선} + d \times \text{높이} + e \times \text{두께} + f$$

$$\phi = \frac{1}{1 + e^{-z}}$$






```
lr = LogisticRegression()
lr.fit(train_bream_smelt, target_bream_smelt)

proba = lr.predict_proba(test_scaled[:5])
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]]

print(lr.coef_.shape, lr.intercept_.shape)
(7, 5) (7,)
```

LogisticRegression의 손실함수와 최적화기법

loss

optimizer

SGDClassifier

손실함수(Loss Function)

- 손실함수는 실제값과 예측값의 차이를 계산하는 함수
  - 모델의 예측이 얼마나 정확한지를 평가하는 지표로, 모델을 최적화하기 위해 사용
    - 학습을 종료시키기 위한 조건 함수
  - 오차가 클수록 손실함수의 값이 크고, 오차가 작을수록 손실함수의 값이 작아진다.
  - 손실함수의 값을 최소화하는 (W, b)를 찾아가는 것이 학습 목표
  - 회귀모델-평균제곱오차 MSE
  - 분류모델-크로스엔트로피
    - 낮은 확률로 예측해서 맞추거나, 높은 확률로 예측해서 틀리는 경우 loss가 커짐
    - 이진분류: binary_crossentropy
    - 다중분류: categorical_crossentropy

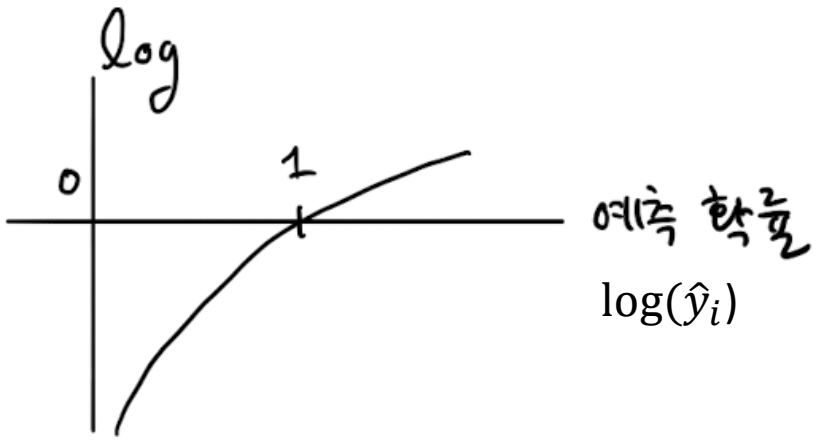
LogisticRegression 손실함수

Logistic Regression의 목표는 주어진 입력 데이터에 대해 클래스에 속할 확률을 예측하는 것.
이 확률이 실제 값과 가까울수록 손실이 적고, 멀어질수록 손실이 커진다.

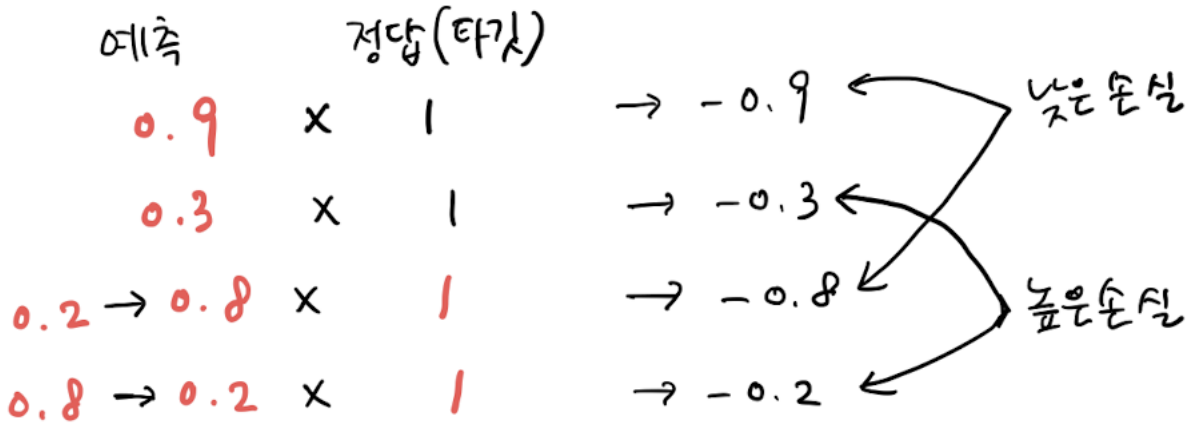
- **이진 분류모델 평가 기준**
  - 로그 손실 함수(Log Loss) 또는 이진 크로스 엔트로피 손실 함수(Binary Cross-Entropy Loss)

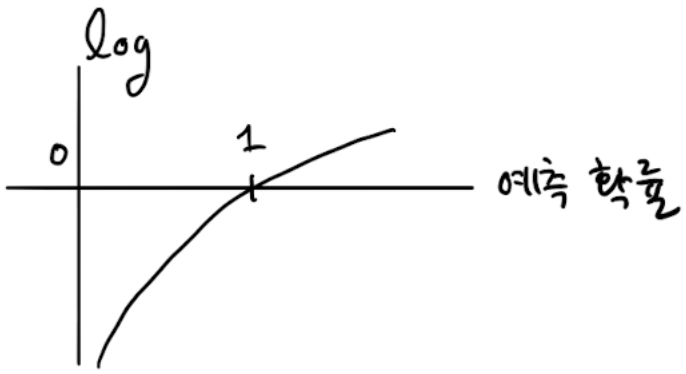
$$\text{Log Loss} = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

- m은 전체 샘플 수
- y_i는 실제 클래스 레이블 (0 또는 1).
- $\hat{y}_i$는 모델이 예측한 확률 (0과 1 사이의 값).



LogisticRegression 손실함수





타깃 = 1 일때

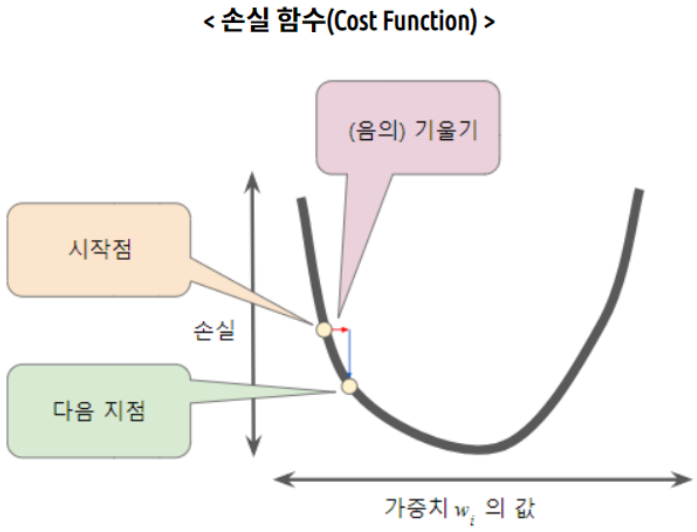
→ - log (예측 확률)

타깃 = 0 일때

→ - log (1 - 예측 확률)

최적화 기법(Optimizer, 옵티마이저)

- 손실함수 결과(손실값)를 최소화 시키는 방법
  - 경사하강법(Gradient Descent)
    - 가장 기본이 되는 옵티마이저 알고리즘
    - 경사를 따라 내려가면서 가중치를 업데이트 시킴
    - 학습률이 너무 크면 학습시간은 짧아지나 전역 최솟값을 찾을 수 없게 됨
    - 학습률이 너무 작으면 학습시간이 오래 걸리고, 지역 최솟값에 수렴할 수 있음
    - 확률적 경사하강법(Stochastic GD)
      - 가중치 업데이터를 전체 데이터가 아니라 랜덤으로 선택한 하나의 데이터에 대해서만 계산함
    - 배치/미니배치 경사하강법
      - 배치: 가중치 등의 매개 변수의 값을 조정하기 위해 사용하는 데이터의 양
      - 배치를 전체 데이터로 둠
      - 1epoch당 시간은 오래 걸리고 메모리를 크게 요구하지만, 전역 최솟값을 찾을 수 있다.
      - 미니배치: 정해진 데이터 양에 대해서만 계산하여 매개변수값 조정
      - 배치보다 빠르고, GD보다 안정적

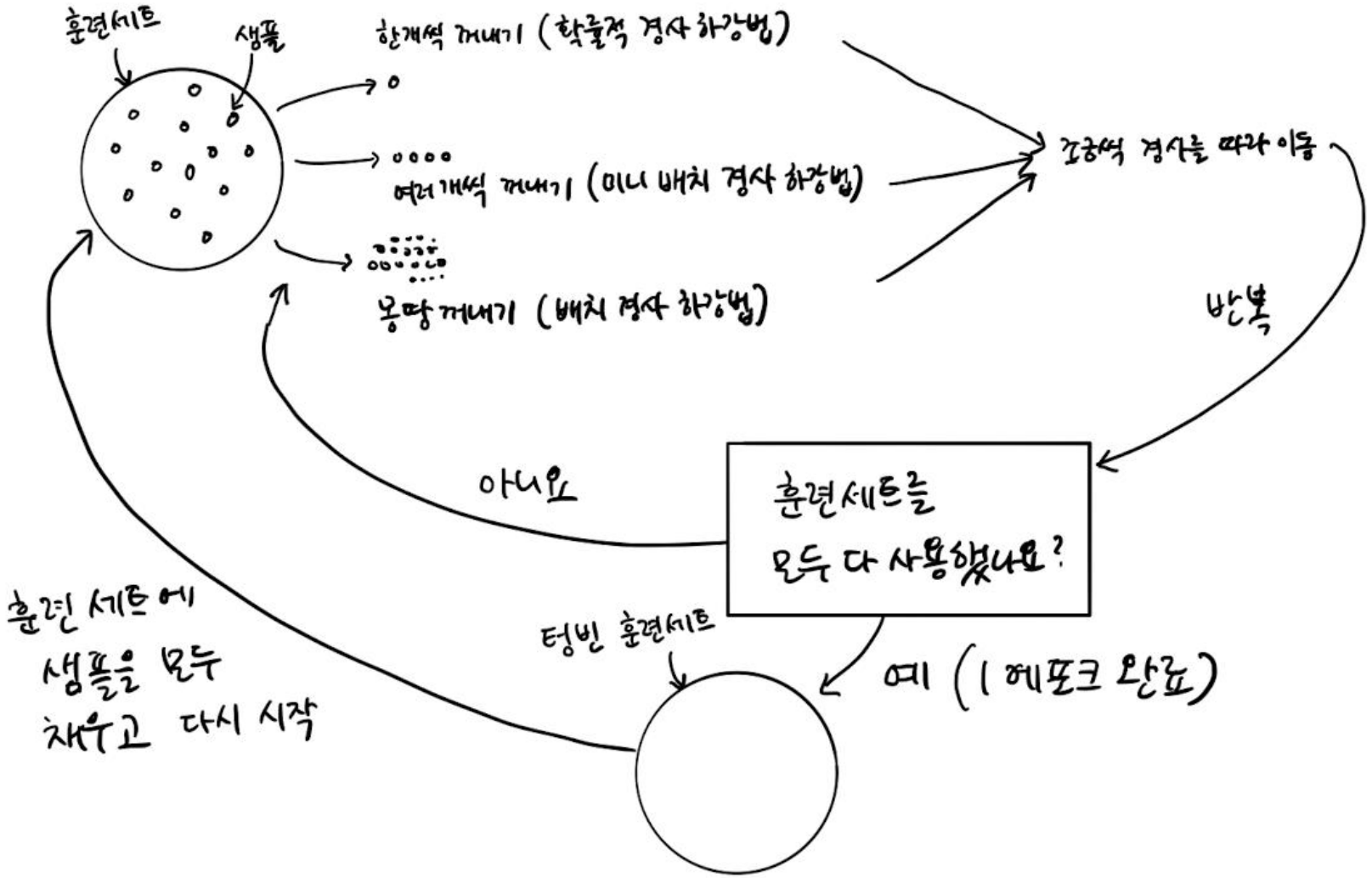


$$w_i = w_i - \eta \frac{\partial f}{\partial w_{i-1}}$$

η: 학습률(learning rate)

최적화 기법(Optimizer, 옵티마이저)

η: 학습률(learning rate)



분류모델: SGDClassifier

- 로그손실함수와 SGD 최적화기법 적용

1. 데이터 전처리

```
import pandas as pd
fish = pd.read_csv('https://bit.ly/fish_csv_data')
fish_input = fish[['Weight','Length','Diagonal','Height','Width']].to_numpy()
fish_target = fish['Species'].to_numpy()
```

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(
    fish_input, fish_target, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

분류 모델: SGDClassifier

```
SGDClassifier(loss='log', max_iter=10, random_state=42)
```

- loss - 손실함수의 종류 지정 (log = 로지스틱 손실 함수)
- max_iter - 수행할 에포크 횟수 지정

SGDClassifier.**partial_fit(train_scaled, train_target)**

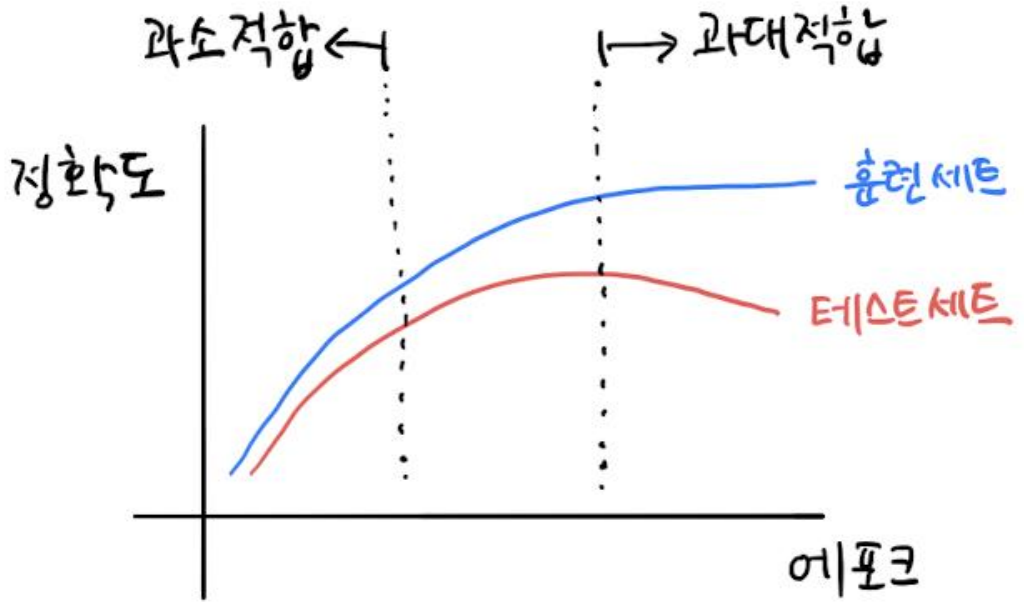
- partial_fit():- 모델을 이어서 훈련할 때 사용.
  :- 호출할 때마다 1 에포크씩 이어서 훈련가능.
  : 적정 에포크 횟수 결정이 중요!!

분류모델: SGDClassifier

```
from sklearn.linear_model import SGDClassifier

sc = SGDClassifier(loss='log', max_iter=10, random_state=42)
sc.fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
0.773109243697479
print(sc.score(test_scaled, test_target))
0.775


sc.partial_fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
0.8151260504201681
print(sc.score(test_scaled, test_target))
0.825
```

분류 모델: SGDClassifier

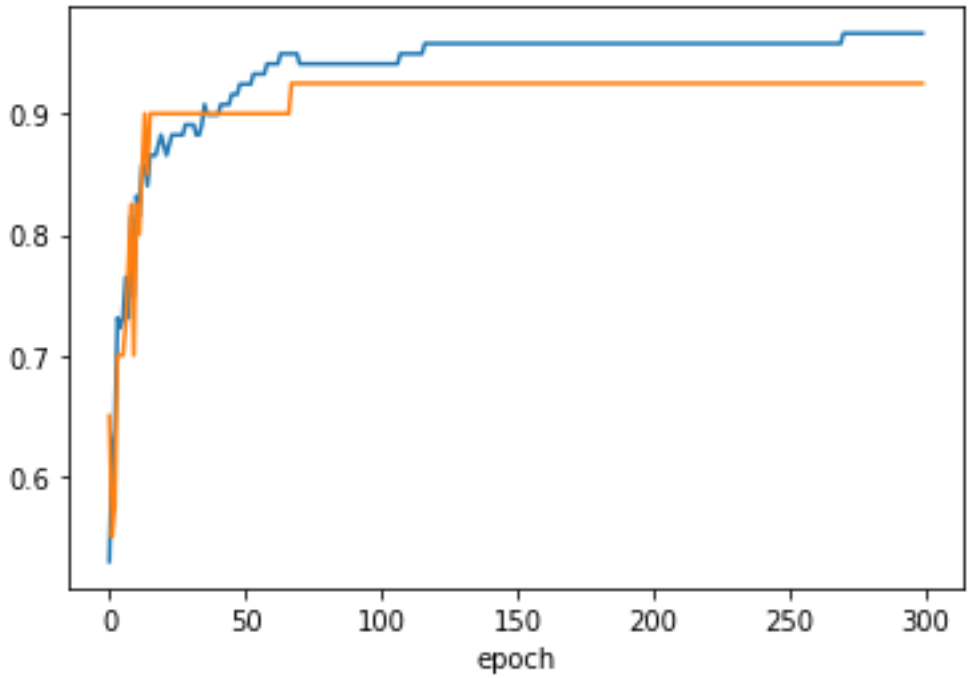
- 에포크와 과대/과소적합



분류 모델: SGDClassifier

- Epoch 횟수 결정

```
sc = SGDClassifier(loss='log', random_state=42)
train_score = []
test_score = []

classes = np.unique(train_target)
for _ in range(0, 300):
    sc.partial_fit(train_scaled, train_target,
                   classes=classes)
    train_score.append(sc.score(train_scaled,
                                train_target))
    test_score.append(sc.score(test_scaled,
                               test_target))
```




```
sc = SGDClassifier(loss='log', max_iter=100,
                   tol=None, random_state=42)
sc.fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
0.957983193277311
print(sc.score(test_scaled, test_target))
0.925
```

• 실습: 최적의 생선 무게 예측 모델 만들기

- SGDRegressor 클래스를 활용한 생선데이터의 무게를 예측하는 모델을 만들기
  생선데이터를 활용
- 손실함수: MSE

SGDRegressor(loss='squared_loss', max_iter=?)

```
import pandas as pd

fish = pd.read_csv('https://bit.ly/fish_csv_data')
fish.head()
```

	Species	Weight	Length	Diagonal	Height	Width
0	Bream	242.0	25.4	30.0	11.5200	4.0200
1	Bream	290.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	26.5	31.1	12.3778	4.6961
3	Bream	363.0	29.0	33.5	12.7300	4.4555
4	Bream	430.0	29.0	34.0	12.4440	5.1340

• 실습: Red, White Wine 분류 모델 만들기

➢ SGDClassifier 를 활용하여 다음 데이터를 가장 잘 분류하는 모델을 만들어 보시오.

SGDClassifier (loss=?', max_iter=?)

```
import pandas as pd

wine = pd.read_csv('https://bit.ly/wine-date')
```

```
wine.describe()
```

	alcohol	sugar	pH	class
count	6497.000000	6497.000000	6497.000000	6497.000000
mean	10.491801	5.443235	3.218501	0.753886
std	1.192712	4.757804	0.160787	0.430779
min	8.000000	0.600000	2.720000	0.000000
25%	9.500000	1.800000	3.110000	1.000000
50%	10.300000	3.000000	3.210000	1.000000
75%	11.300000	8.100000	3.320000	1.000000
max	14.900000	65.800000	4.010000	1.000000