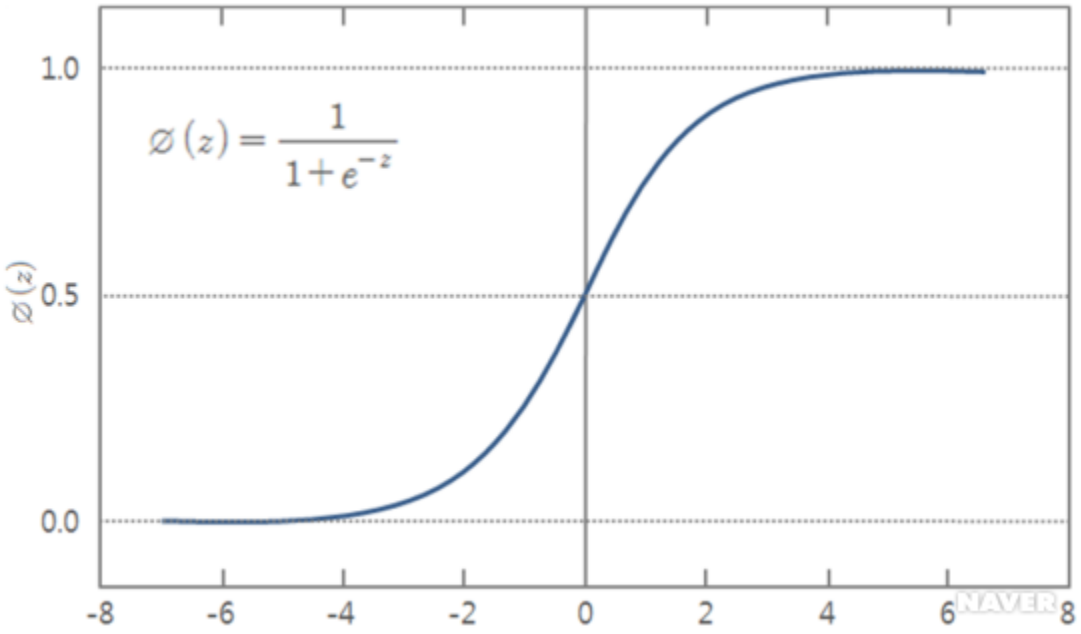
ch4

로지스틱회귀

이름은 회귀 이지만 분류모델의 한종류
선형회귀와 동일하게 선형 방정식을 학습한다.
이진 분류 & 다중 분류

시그모이드 (로지스틱함수)

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



로지스틱함수 사용법

사이킷런에 있기 때문에 그냥 불러와서 사용(이진분류해보기)

```
bream_smelt_indexes = (train_target == 'Bream') | (train_target == 'Smelt')
#이진분류를 위해서 Bream(도미) 와 # Smelt(빙어)를 빼온다
train_bream_smelt = train_scaled[bream_smelt_indexes]
target_bream_smelt = train_target[bream_smelt_indexes]
#훈련+테스트 케이스 만들기

from sklearn.linear_model import LogisticRegression
# LogisticRegression 가 바로 로지스틱 회귀를 가지고 오는것이다.
lr = LogisticRegression()
lr.fit(train_bream_smelt, target_bream_smelt)
#그냥 이렇게 써주면 끝!
```

```
print(lr.predict(train_bream_smelt[:5]))
#5개의 샘플 출력을 예측
```

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

```
print(lr.predict_proba(train_bream_smelt[:5]))
# lr.predict_proba 사용해서 예측확률을 출력
```

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

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

```
print(lr.classes_)
```

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

```
print(lr.coef_, lr.intercept_)
#coef는 계수 #intercept 는 상수를 나타냄
```

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

Z값 출력하기

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

```
print(decisions)
```

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

시그모이드 함수 사용하기

```
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]]
```

양성 클래스에 대한
z값을 반환

다중 분류 수행하기
7마리의 생선으로 해보기

```
lr = LogisticRegression(C=20, max_iter=1000)
# C는 LogisticRegression 함수를 규제하는 매개변수이다.
#L2 규제라고 하고 이는 기본적으로 릿지회귀와 같이 계수의 제곱을 규제함
#단 alpha는 숫자가 클수록 규제가 커진다면 c는 작을수록 규제가 커짐
#LogisticRegression에서 c의 기본값은 1이므로 여기서는 규제를 완화하기위해 20을 넣어줄거임
#잠깐 복습 규제란 훈련세트를 너무 과도하게 학습하지 못하도록 훼방하는 것을 말함(퀴즈 훈련세트가 과도하게 학습되는걸 뭐라고 할까요?)
#max_iter=1000는 1000번 정도반복하라는 뜻
lr.fit(train_scaled, train_target)

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

```
0.9327731092436975
0.925
```

```
1  print(lr.predict(test_scaled[:5]))
2  #예측해보기
```

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

```
1  proba = lr.predict_proba(test_scaled[:5])
2  print(np.round(proba, decimals=3))
3  #decimals=3 소숫점 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]]
```

```
1  print(lr.classes_)
```

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

Coef와 intercept는 어떻게나올까?

```
1  print(lr.coef_.shape, lr.intercept_.shape)
2
```

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

8행으로 shape 변환

```
arr.reshape(8,)
array([1, 2, 3, 4, 5, 6, 7, 8])
```

4행 2열의 shape 변환

```
arr.reshape(4,2)
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

즉 coef는 열5개 행은 7개

Intercept도 7개 이므로

Z값이 7개가 나온다.=> 소프트 맥스 함수로 해결

$$S1 = \frac{e^{z1}}{e\_sum}, S2 = \frac{e^{z2}}{e\_sum}, \cdots, S7 = \frac{e^{z7}}{e\_sum}$$

소프트맥스(softmax) 함수

분류를 위한 출력층 함수로 0~1 사이의 숫자를 출력하는 함수입니다.
공식은 다음과 같습니다.

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

```
[[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]]
```

```
1  decision = lr.decision_function(test_scaled[:5])
2  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]]
```

```
1  from scipy.special import softmax
2
3  proba = softmax(decision, axis=1)
4  #axis 1은 각행마다 소프트맥스 계산한다는 뜻~
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]]
```

확률적 경사 하강법

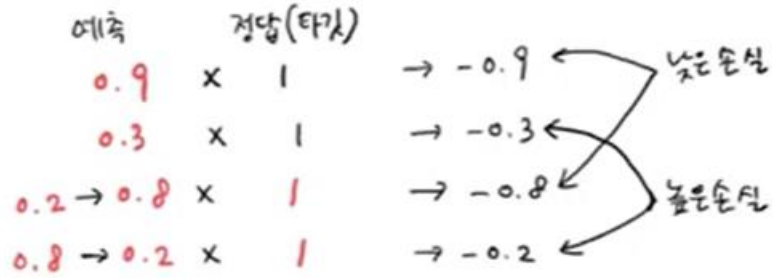
- 확률적 경사 하강법은 훈련세트에서 랜덤하게 하나의 샘플을 선택하여 손실함수의 경사를 따라 최적의 모델을 찾는 알고리즘이다.

- 모델의 예측값과 실제값 사이의 오차를 정량화하는 함수로

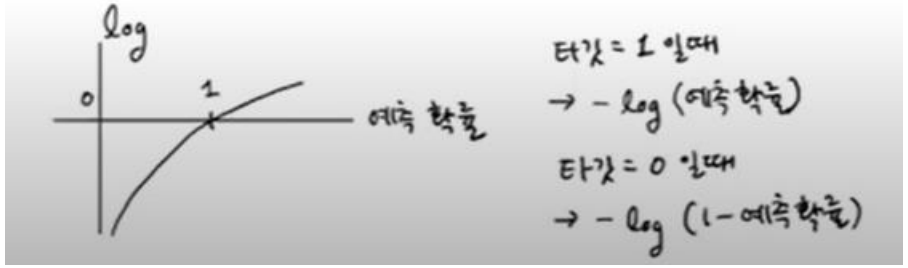
- 예측확률과 손실함수는 반비례 관계
- 만약 다 못내려가면 훈련세트에 다시 샘플을 넣어서 실행
- 이때 한번의 훈련세트를 다쓰는걸 '에포크' 라고함
- 한개씩: 확률적 경사 하강법 , 무작위로 몇개씩: 미니배치 경사 하강법
- 전체샘플: 배치 경사 하강법

손실함수

- 어떤 문제에서 머신러닝 알고리즘이 얼마나 엉터리 인지를 측정정하는 기준, 작을수록 NO 엉터리이다.
- 분류에서 손실은 정답을 못 맞추면 손실이다.
- 손실함수는 연속적이여야 한다.
- 예측에 –를 붙여줘서 커질수록 손실이 커짐을 뜻한다.



- 오른쪽 식을 이지로지스틱 손실함수라한다.
- 다중은 크로스엔트로피 손실 함수라고한다.
- 사실 그냥 이런게 있다~ 만 알고 넘어가도
- 머신러닝 라이브러리가 다해줍니다!



```
import pandas as pd

fish = pd.read_csv('https://bit.ly/fish_csv_data')
#판다스를 이용한 데이터 가지고오기
```

```
fish_input = fish[['Weight','Length','Diagonal','Height','Width']].to_numpy()
fish_target = fish['Species'].to_numpy()
#Species 제외한 5가지 특성으로 입력 데이터 만들기
```

```
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)
#사이킷런을 사용한 훈련세트와 테스트 세트만들기
```

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
#표준화 처리
```

```
from sklearn.linear_model import SGDClassifier
```

```
sc = SGDClassifier(loss='log_loss', max_iter=10, random_state=42)
#두개의 매개변수를 정함 loss는 손실함수 max_iter는 에포크
sc.fit(train_scaled, train_target)

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

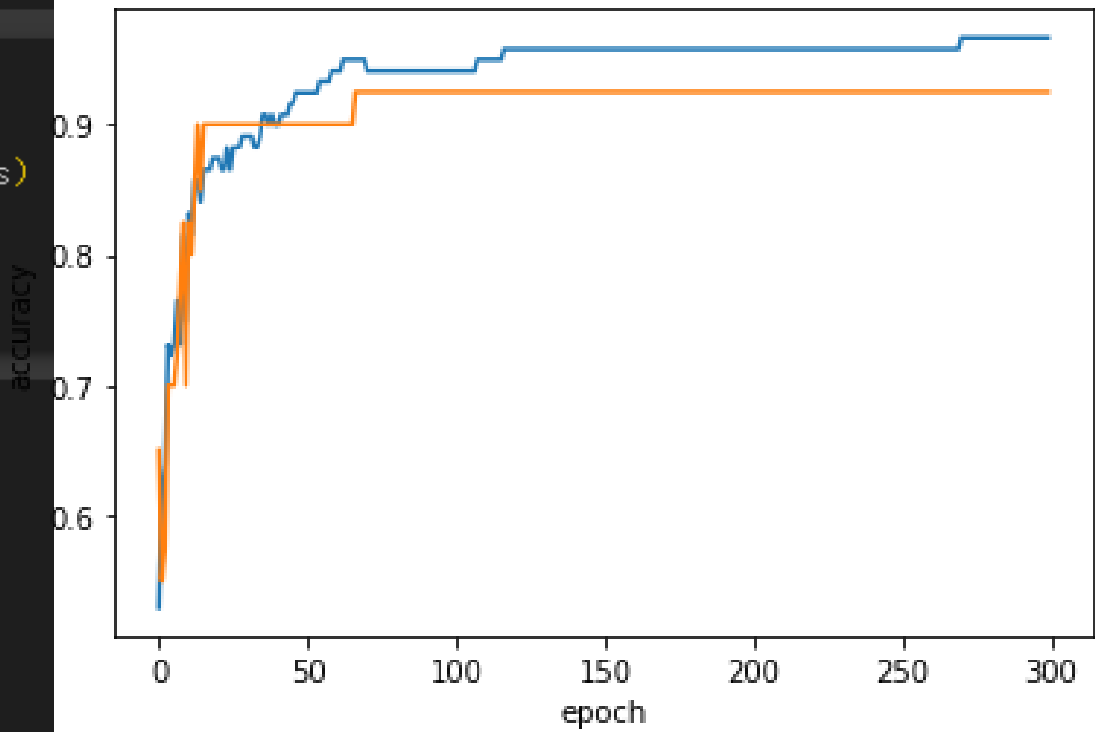
```
0.7731092436974789
0.775
```

- 훈련과 테스트 모두 낮다 에포크를 조절하면 될 것 같은데
- 그렇다면 얼마나 조절을 해야할까??

```python
import numpy as np

sc = SGDClassifier(loss='log_loss', random_state=42)

train_score = []
test_score = []
#에포크마다 두변화를 측정하기 위해 두개의 리스트 준비

classes = np.unique(train_target)
#np.unique 함수를 통해 7개의 생성 목록생성
```

```python
for _ in range(0, 300):
    #300번의 에포크 진행해보기
    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))
```

```python
import matplotlib.pyplot as plt

plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```



- 최적의 에포크는 100이었다 대입해보자

```
1   sc = SGDClassifier(loss='log_loss', max_iter=100, tol=None, random_state=42)
2   sc.fit(train_scaled, train_target)
3
4   print(sc.score(train_scaled, train_target))
5   print(sc.score(test_scaled, test_target))
```

```
0.957983193277311
0.925
```