

로지스틱 회귀

- 선형 방정식을 사용한 분류 알고리즘

```
In [1]: # 데이터 준비하기

import pandas as pd
fish=pd.read_csv("fish.csv")
fish.head()
```

```
Out[1]:
```

	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

```
In [2]: print(pd.unique(fish["Species"])) # 열의 고유한 값 추출
['Bream' 'Roach' 'Whitefish' 'Parkki' 'Perch' 'Pike' 'Smelt']

In [3]: fish_input=fish[["Weight","Length","Diagonal","Height","Width"]].to_numpy()
```

```
In [4]: print(fish_input[:5])

[[242.    25.4    30.     11.52    4.02   ]
 [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]
 [430.    29.     34.     12.444   5.134   ]]
```

```
In [5]: fish_target=fish["Species"].to_numpy()
```

```
In [6]: 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)
```

```
In [7]: from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss.fit(train_input)
train_scaled=ss.transform(train_input)
test_scaled=ss.transform(test_input)
```

k-최근접 이웃 분류기의 확률 예측

- 다중 분류: 타깃 데이터에 2개 이상의 클래스가 포함된 문제
- n_neighbors의 (개수+1)만큼의 확률만 나온다는 단점이 있음. (ex) 3일 때 -> 0/3,1/3,2/3,3/3

```
In [8]: 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))

0.8907563025210085
0.85

In [9]: print(kn.classes_) # 사이킷런 모델에서는 타깃값이 알파벳 순으로 매겨짐.
['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']

In [10]: print(kn.predict(test_scaled[:5]))
['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']

In [11]: import numpy as np
proba=kn.predict_proba(test_scaled[:5]) # predict_proba 함수로 확률을 출력할 수 있음
print(np.around(proba,decimals=4))

[[0.    0.    1.    0.    0.    0.    ]
 [0.    0.    0.    0.    0.    1.    ]
 [0.    0.    0.    1.    0.    0.    ]
 [0.    0.    0.6667 0.    0.3333 0.    ]
 [0.    0.    0.6667 0.    0.3333 0.    ]]
```

```
In [12]: # 이 모델이 계산한 확률이 가장 가까운 이웃의 비율이 맞는지 직접 확인
distances,indexes=kn.kneighbors(test_scaled[3:4])
print(train_target[indexes])

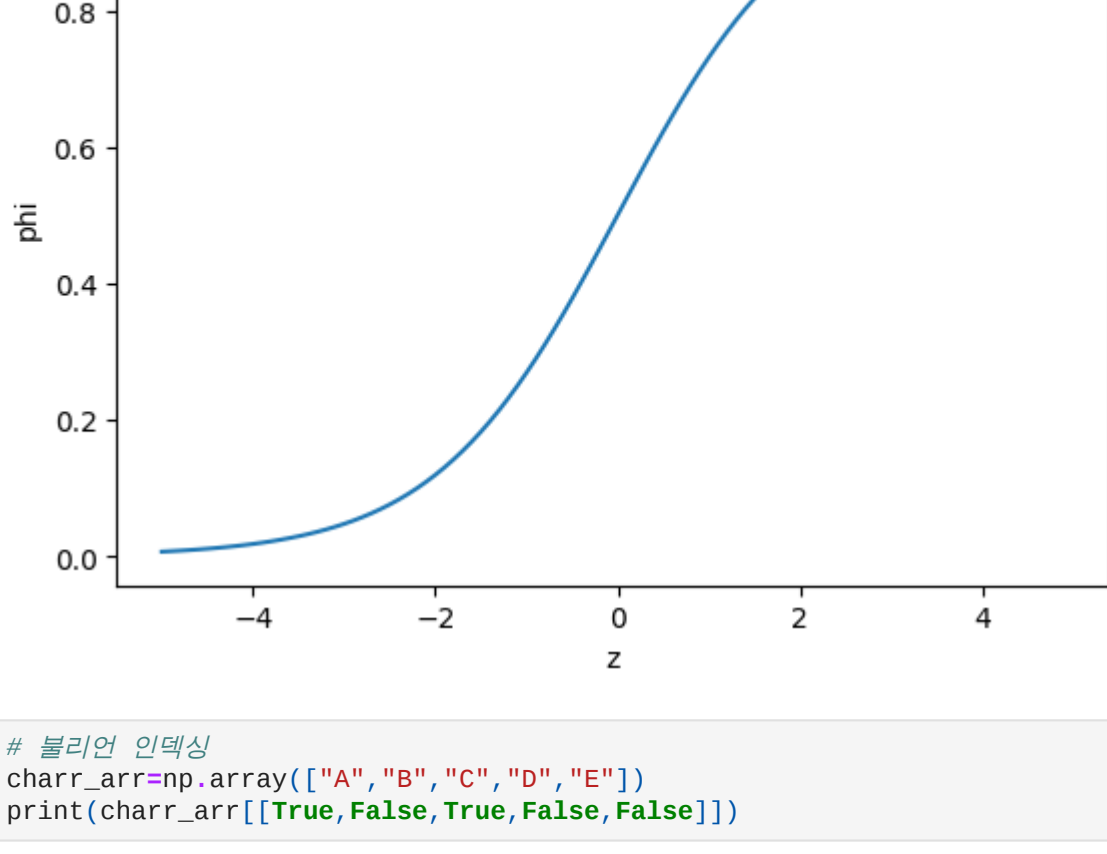
# 성공임

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

로지스틱 회귀

- 이들은 회귀이지만 분류 모델임
- 선형 회귀와 동일하게 선형 방정식을 학습함
- (ex) $z = a \times (Weight) + b \times (Length) + c \times (Diagonal) + d \times (Height) + e \times (Width) + f$
- 시그모이드 함수(로지스틱 함수)
 - 확률은 0~1이기 때문에 z가 아주 큰 음수일 때 0이 되고, 큰 양수일 때 1이 되도록 바꾸기 위함
 - $\phi = \frac{1}{1+e^{-z}}$
- 이진 분류 시 시그모이드 함수의 출력값이 0.5보다 크면 양성, 0.5보다 작거나 같으면 음성 클래스로 판단함

```
In [13]: 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()
```



```
In [14]: # 불리언 인덱싱
charr_arr=np.array(["A","B","C","D","E"])
print(charr_arr[[True,False,True,False,False]])
['A' 'C']

In [15]: 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]
```

```
In [16]: from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(train_bream_smelt,target_bream_smelt)
```

```
Out[16]: LogisticRegression
LogisticRegression()
```

```
In [17]: print(lr.predict(train_bream_smelt[:5]))
['Bream' 'Smelt' 'Bream' 'Bream' 'Bream']

In [18]: # 첫번째 열이 음성 클래스(0), 두 번째 열이 양성 클래스(1)에 대한 확률임
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]]

In [19]: print(lr.classes_)
['Bream' 'Smelt']

In [20]: print(lr.coef_,lr.intercept_)
[[-0.4037798  -0.57620209 -0.66280298 -1.01290277 -0.73168947]] [-2.16155132]

In [21]: decisions=lr.decision_function(train_bream_smelt[:5]) # decision_function() 메서드로 양성 클래스의 z 값을 출력함
print(decisions)
[-6.02927744  3.57123907 -5.26568906 -4.24321775 -6.0607117 ]

In [22]: # z값을 통과시켜 확률을 얻을 수 있음
# predict_proba()와의 두 번째 열의 값과 동일함
from scipy.special import expit
print(expit(decisions))

[0.00240145 0.97264817 0.00513928 0.01415798 0.00232731]
```

로지스틱 회귀로 다중 분류 수행하기

- 로지스틱 회귀 클래스는 기본적으로 반복적인 알고리즘을 사용함
- max_iter 매개변수에서 반복 횟수를 지정하여 기본값은 100임
- 핏치 회귀와 같이 계수의 제곱을 규제함
- 그러나 alpha 매개변수가 아니라 C 매개변수를 사용함. alpha와 다르게 C가 작을수록 규제가 큼. C의 기본값은 1임
- 이중 분류와 달리 시그모이드 함수가 아니라, 소프트맥스 함수를 사용하여 z값을 확률로 변환함
- 소프트맥스 함수
 - $esum = e^{z1} + e^{z2} + \dots$
 - $s1 = e^{z1}/esum, s2 = e^{z2}/esum \dots$

```
In [23]: 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

In [24]: print(lr.predict(test_scaled[:5]))
['Perch' 'Smelt' 'Pike' 'Roach' 'Perch']

In [25]: 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]]

In [26]: print(lr.classes_)
['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']

In [27]: print(lr.coef_.shape,lr.intercept_.shape) # 다중 분류는 클래스마다 z값을 하나씩 계산함
(7, 5) (7,)
```

```
In [28]: 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]]

In [29]: 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]]
```

확률적 경사 하강법

- 확률적: 무작위하게 혹은 랜덤하게. 훈련 세트를 한번에 전부 사용하지 않고 랜덤하게 골라서 사용함
- 경사 하강법: 강사를 따라 내려가는 방법. 가장 가까운 강사를 따라 원하는 지점에 도달하는 것이 목표
- 가장 가까운 길을 찾아 내려오지만 조금씩 내려오는 것이 중요함
- 훈련 세트에서 랜덤하게 하나의 샘플을 선택하여 가파른 강사를 조금 내려감. 이 과정을 전체 샘플을 모두 사용할 때까지 반복
- 강사를 다 내려오지 못했으면 다시 처음부터 시작함
- 에포크: 확률적 경사 하강법에서 훈련 세트를 한 번 모두 사용하는 과정
- 미니배치 경사 하강법: 1개씩 말고 무작위로 몇개의 샘플을 선택해서 강사를 따라 내려가는 방식
- 배치 경사 하강법: 한 번 강사를 따라 이동하기 위해 전체 샘플을 사용하는 방식. 가장 안정적일 수 있지만 매우 비효율적일 수 있음

점진적인 학습

- 훈련 데이터가 한 번에 준비되는 것이 아니라 조금씩 전달됨
- 확률적 경사 하강법이 대표적인 점진적 학습 알고리즘임

손실 함수

- 어떤 문제에서 머신러닝 알고리즘이 얼마나 엉터리인지 측정하는 기준
- 작을수록 좋지만, 어떤 값이 최솟값인지 모르기 때문에 만족할만한 수준이면 산을 다 내려왔다고 인정함
- 경사 하강법을 사용할 때 아주 조금씩 내려와야하기 때문에 손실 함수의 값이 연속적이어야 함 -> 정확도를 사용하기 어려움. 확률을 사용함

로지스틱 손실 함수(이진 크로스엔트로피 손실 함수)

- 타깃이 0일 경우, 예측에 1을 곱한 값에 음수를 취함
- 타깃이 0일 경우, (1-예측)에 1을 곱한 값에 음수를 취함
- 위의 예측 확률을 로그함수를 적용하여 더 좋음. 양수를 얻기 위함임
- 크로스엔트로피 손실 함수: 다중 분류에서 사용하는 손실 함수

```
In [30]: import pandas as pd
fish=pd.read_csv("fish.csv")
fish_input=fish[["Weight","Length","Diagonal","Height","Width"]].to_numpy()
fish_target=fish["Species"].to_numpy()
```

```
In [31]: 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)
```

```
In [32]: from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss.fit(train_input)
train_scaled=ss.transform(train_input)
test_scaled=ss.transform(test_input)
```

```
In [33]: from sklearn.linear_model import SGDClassifier
sc=SGDClassifier(loss="log",max_iter=10,random_state=42) # loss="log" 는 로지스틱 손실 함수를 지정하는 것, max_iter은 에포크 횟수
sc.fit(train_scaled,train_target)
print(sc.score(train_scaled,train_target))
print(sc.score(test_scaled,test_target))

0.773109243697479
0.775

C:\Users\82106\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be removed in version 1.3. Use 'loss='log_loss'' which is equivalent.
warnings.warn(
C:\Users\82106\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:702: ConvergenceWarning: Maximum number of iteration reached before co
nvergence. Consider increasing max_iter to improve the fit.
warnings.warn(
```

```
In [34]: # 점진적 학습
sc.partial_fit(train_scaled,train_target)
print(sc.score(train_scaled,train_target))
print(sc.score(test_scaled,test_target))

0.8151260504201681
0.85
```

에포크와 과대/과소 적합

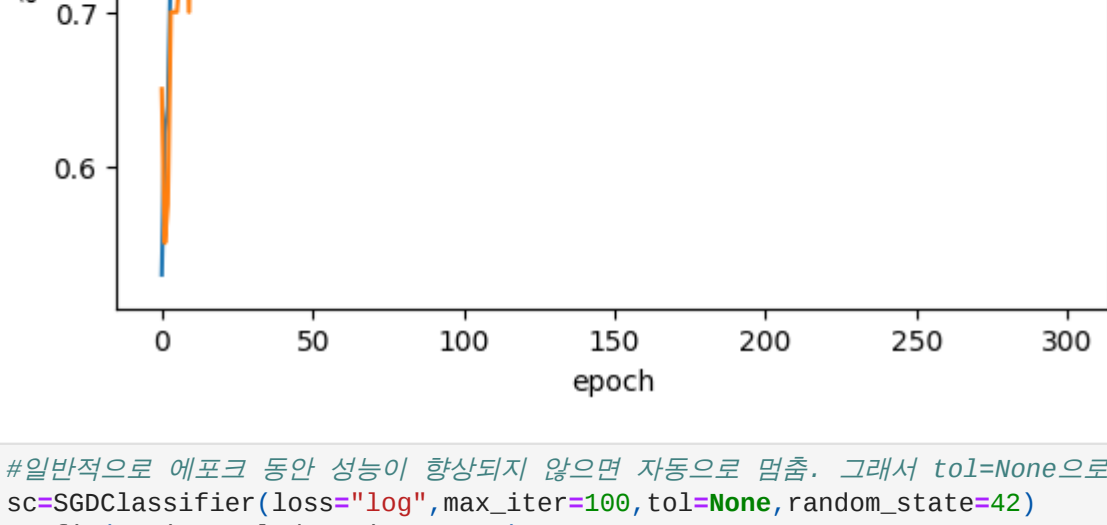
- 일반적으로 에포크 횟수가 적으면 과소적합될 수 있음
- 일반적으로 에포크 횟수가 많으면 과대적합될 수 있음
- 조기 종료: 과대적합이 시작하기 전에 훈련을 멈추는 것

```
In [35]: import numpy as np
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) # partial_fit 메서드만 사용하기 위해 classes 매개변수를 사용함
    train_score.append(sc.score(train_scaled,train_target))
    test_score.append(sc.score(test_scaled,test_target))

import matplotlib.pyplot as plt

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



```
In [36]: #일반적으로 에포크 동안 성능이 향상되지 않으면 자동으로 멈춤. 그래서 tol=None으로 안 멈추게 함
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))
print(sc.score(test_scaled,test_target))

0.957983193277311
0.925

C:\Users\82106\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:163: FutureWarning: The loss 'log' was deprecated in v1.1 and will be removed in version 1.3. Use 'loss='log_loss'' which is equivalent.
warnings.warn(
```

- loss 매개변수의 기본 값은 hinge(힌지 손실, 서포트 벡터 머신)임