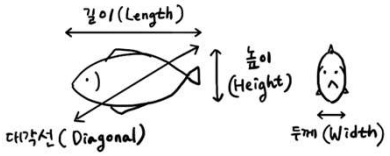
로지스틱 회귀로 클래스 확률 구하기!



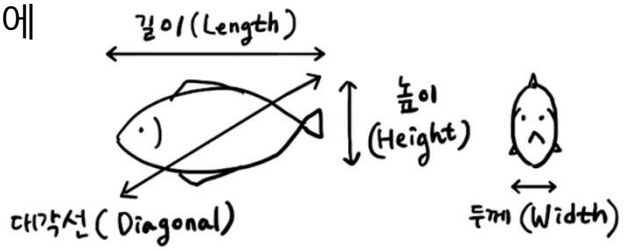
도미일 확률 : 72%
빙어일 확률 : 16%
⋮

내가 가진 데이터로 어떤 생선이 나올지 확인해 보자!

1

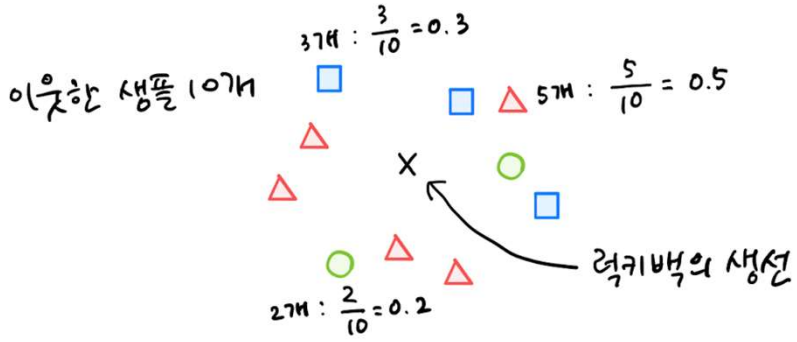
목표 : 내가 가진 데이터로 생선 종류의 확률 구하기

- 총 7가지 생선 존재 가정
- 생선의 크기, 무게 등이 주어 졌을 때 7가지 생선에 대한 확률을 출력 하는 것이 목표!
- 이를 위해 길이, 높이, 두께 이외에 대각선 길이, 무게 데이터 사용



2

확률 계산하기 : K-최근접 이웃 경우



3

데이터 준비

```
import pandas as pd

fish = pd.read_csv('https://raw.githubusercontent.com/Gonteer/2024DongALINC/main/001ML/00104_Regression_Logistic/fish.csv')
fish.head()
```

1.6s

	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

```
print(pd.unique(fish['Species']))
```

0.0s

#해당 열의 고유 값 추출 후 확인
→ 데이터에서 어떤 생선이 있는지 확인

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

4

데이터 준비

#선택 필드 numpy 배열 변환

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

[4] ✓ 0.0s

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

[5] ✓ 0.0s

```
[[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 ]]
```

#numpy 배열 변환 후 상위 5개 데이터 확인

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

#타깃 데이터 생성 → 7가지의 생선 이름 저장

[6] ✓ 0.0s

5

데이터 준비

```
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)
```

데이터 세트 준비

✓ 2.1s

```
from sklearn.preprocessing import StandardScaler

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

표준화 전처리 실시

✓ 0.0s

6

k-최근접 이웃의 다중 분류

```
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))
```

✓ 0.2s

```
0.8907563025210085
0.85
```

#클래스 확률을 구하는 것이라 데이터 세트 점수 중요하지 않음

7

k-최근접 이웃의 다중 분류

```
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))
```

✓ 0.2s

```
0.8907563025210085
0.85
```

#클래스 확률을 구하는 것이라
데이터 세트 점수 중요하지 않음

8

k-최근접 이웃의 다중 분류

```
print(kn.classes_)
```

#구별할 클래스 종류 확인
→ 여기선 7가지 생선 Class 출력(알파벳 오름차순 정렬)

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

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

#샘플 데이터 중 상위 5개 데이터 이용해 예측 확인

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

9

k-최근접 이웃의 다중 분류

```
import numpy as np

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

#predict_proba()메서드로 클래스 확률 값 변환(상위 5개 데이터 예측 값만)

#round함수 사용해 소수점 넷째자리에서 반올림해 출력

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

첫번째 클래스(Bream)에 대한 확률

첫번째 샘플 → [0, 0, 0.6667, 0, 0.3333, 0, 0]

두번째 클래스(Parkki)에 대한 확률

```
distances, indexes = kn.kneighbors(test_scaled[3:4])
print(train_target[indexes])
```

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

10

여기서 이상적인 방법 → 로지스틱 회귀!

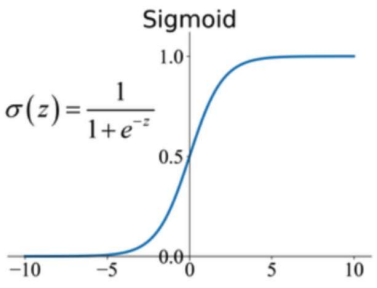
- 로지스틱 회귀는 '분류 모델'
  - 선형 회귀와 동일하게 선형 방정식 학습

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

  - a,b,c,d,e 가중치 (또는 계수) → 특성이 늘어났지만 다중회귀를 위한 선형 방정식과 동일
  - z는 어떤 값도 가능하지만 확률이 되려면 0~1(0~100%) 사이 값이 되어야 함

11

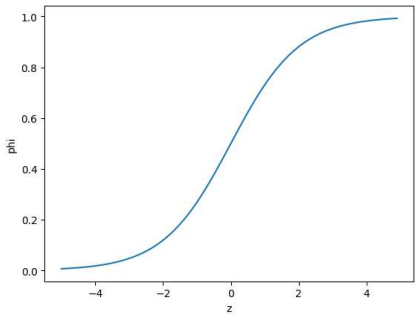
시그모이드(로지스틱) 함수



- 선형방정식의 출력 z의 음수를 이용해 자연상수 e를 거듭제곱하고 1을 더한 역수를 취함
  - z가 무한하게 큰 음수일 경우 함수는 0에 가까워 짐
  - z가 무한하게 큰 양수일 경우 함수는 1에 가까워 짐
  - z가 0일 때 0.5가 됨
- Z가 어떤 값이 되더라도 σ(S)는 0~1 사이의 범위를 벗어나지 않음

12

시그모이드(로지스틱) 함수




```
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()
```

0.6s

사이킷런에서 'LogisticRegression' 클래스 제공!

13

로지스틱 회귀(이진 분류)

- 일단 도미(Bream)와 빙어(Smelt) 데이터만 추출함

```
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]
```

0.0s

- bream_smelt_indexes 배열에선 Bream과 Smelt는 True, 나머진 False 값 부여
- 이후 test 와 target 데이터 셋은 Boolean index 실시 (밑에 예시)

```
char_arr = np.array(['A', 'B', 'C', 'D', 'E'])
print(char_arr[[True, False, True, False, False]])
```

```
['A' 'C']
```

14

로지스틱 회귀(이진 분류)

- LogisticRegresssion 모델로 학습

```
from sklearn.linear_model import LogisticRegression

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

✓ 0.0s

- 훈련된 모델로 train_bream_smelt에 있는 상위 5개 샘플 예측

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

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

15

로지스틱 회귀(이진 분류)

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

✓ 0.0s

```
[[0.99760007 0.00239993]
 [0.02737325 0.97262675]
 [0.99486386 0.00513614]
 [0.98585047 0.01414953]
 [0.99767419 0.00232581]]
```

#predict_proba()메서드로 클래스 확률 값 변환(상위 5개 데이터 예측 값만)

결과에서

첫번째 열이 음성(0) 클래스,
두번째 열은 양성(1) 클래스 임

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

✓ 0.0s

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

#어떤 클래스가 양성 클래스 인지 확인

Bream이 음성(0), Smelt이 양성(1)

16

로지스틱 회귀 계수 확인

```
print(lr.coef_, lr.intercept_)
```
#학습한 계수 확인

```
[[-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)
```
#decision_function()메소드로 z값 출력

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

```
from scipy.special import expit

print(expit(decisions))
```
✓ 0.0s

#scipy의 expit()함수를 이용해 decisions 배열 값을 시그모이드 함수를 이용해 확률 값으로 변환

결과 값이 양성(1) 클래스 기점 계산 됨

```
[0.00239993 0.97262675 0.00513614 0.01414953 0.00232581]
```

17

로지스틱 회귀(다중 분류)

일곱 종류의 생선을 분류해 보면서 이진 분류와 차이점 확인

```
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))
```
✓ 0.0s

```
0.9327731092436975
0.925
```

훈련세트와 테스트 세트에 대한 점수가 높음
과대적합/과소적합으로 치우치지 않음

- LogisticRegression 클래스
  - **기본적으로 반복적인 알고리즘 사용**
  - max_iter 매개변수에는 최대 반복횟수 지정 (기본값 100)
  - **계수의 제곱을 규제 필요(L2)**
  - C 매개변수는 규제를 제어하는 값 지정 (기본 값 1)
  - C 값은 작을수록 규제가 커짐

18

로지스틱 회귀(다중 분류)

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

0.0s

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

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

0.0s

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

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

0.0s

```
[[0.    0.014 0.842 0.    0.135 0.007 0.003]
 [0.    0.003 0.044 0.    0.007 0.946 0.   ]
 [0.    0.    0.034 0.934 0.015 0.016 0.   ]
 [0.011 0.034 0.305 0.006 0.567 0.    0.076]
 [0.    0.    0.904 0.002 0.089 0.002 0.001]]
```

19

로지스틱 회귀(다중 분류)

```
print(lr.coef_.shape, lr.intercept_.shape)    #학습한 계수 확인
```

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

- 배열의 열(lr.coef)은 5개 → 5개의 특성 사용
- 배열의 행은 7개 → z를 7개 계산,
  다중분류는 클래스마다 z값 하나씩 계산
  그 중 가장 높은 값을 가진 z가 예측 클래스
- 다중 분류는 소프트맥스(softmax) 함수를 사용해 7개의 z값을 확률로 변환

20

로지스틱 회귀(다중 분류) : Sofmax 함수

- 7개의 z값의 이름을 $z^1 \sim z^7$으로 지정
- $z^1 \sim z^7$값을 이용해 $e^{z1} \sim e^{z7}$ 을 지수함수 을 계산해 모두 더함

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

- $e^{z1} \sim e^{z7}$ 을 각각 e_sum으로 나눔

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

- s1~s7 모두 더하면 분자와 분모가 같아져 1이 됨
- 즉, 7가지 생선(클래스)에 대한 확률의 합은 1이 되야 됨

21

로지스틱 회귀(다중 분류)

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

0.0s

```
[[ -6.51   1.04   5.17  -2.76   3.34   0.35  -0.63]
 [-10.88   1.94   4.78  -2.42   2.99   7.84  -4.25]
 [ -4.34  -6.24   3.17   6.48   2.36   2.43  -3.87]
 [ -0.69   0.45   2.64  -1.21   3.26  -5.7    1.26]
 [ -6.4   -1.99   5.82  -0.13   3.5   -0.09  -0.7 ]]
```

```
from scipy.special import softmax

proba = softmax(decision, axis=1)
print(np.round(proba, decimals=3))
```

0.0s

```
[[0.    0.014 0.842 0.    0.135 0.007 0.003]
 [0.    0.003 0.044 0.    0.007 0.946 0.   ]
 [0.    0.    0.034 0.934 0.015 0.016 0.   ]
 [0.011 0.034 0.305 0.006 0.567 0.    0.076]
 [0.    0.    0.904 0.002 0.089 0.002 0.001]]
```

- softmax 함수
  - **axis** 매개변수엔 소프트맥스로 계산할 축을 지정해야 함
  - axis= 1으로 지정해 각 행(각 샘플)에 대해 계산
  - 만약 axis 지정 안할 경우 배열 전체에 대해 계산

22

감사합니다

내용 출처 정보 : https://www.hanbit.co.kr/store/books/look.php?p_code=B2002963743

23