CHAPTER 04 다양한 분류 알고리즘

혼자 공부하는 머신러닝+딥러닝 (개정판)

한국공학대학교 게임공학과
이재영

학습 로드맵

머신러닝편 **01~06장**

딥러닝만 먼저 배우고
싶다면 01~04장을 읽은 후
07장으로 건너뛰어도 좋습니다.

딥러닝편 **07~10장**

07장을 읽은 후 08장과 09장은
순서대로 읽지 않아도 괜찮습니다. 10
장을 읽기 전에 07장과 09장을
읽는 것이 좋습니다.

난이도

START

01 나의 첫 머신러닝

02 데이터 다루기

03 회귀 알고리즘과 모델 규제

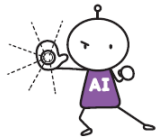
2번 보기

04 다양한 분류 알고리즘

05 트리 알고리즘

06 비지도 학습

07 딥러닝을 시작합니다



08 이미지를 위한 인공 신경망

09 텍스트를 위한 인공 신경망

10 언어 모델을 위한 신경망

GOAL

이 책의 학습 목표

- CHAPTER 01: 나의 첫 머신러닝
  - 인공지능, 머신러닝, 딥러닝의 차이점을 이해합니다.
  - 구글 코랩 사용법을 배웁니다.
  - 첫 번째 머신러닝 프로그램을 만들고 머신러닝의 기본 작동 원리를 이해합니다.
- CHAPTER 02: 데이터 다루기
  - 머신러닝 알고리즘에 주입할 데이터를 준비하는 방법을 배웁니다.
  - 데이터 형태가 알고리즘에 미치는 영향을 이해합니다.
- CHAPTER 03: 회귀 알고리즘과 모델 규제
  - 지도 학습 알고리즘의 한 종류인 회귀 알고리즘에 대해 배웁니다.
  - 다양한 선형 회귀 알고리즘의 장단점을 이해합니다.
- CHAPTER 04: 다양한 분류 알고리즘
  - 로지스틱 회귀, 확률적 경사 하강법과 같은 분류 알고리즘을 배웁니다.
  - 이진 분류와 다중 분류의 차이를 이해하고 클래스별 확률을 예측합니다.
- CHAPTER 05: 트리 알고리즘
  - 성능이 좋고 이해하기 쉬운 트리 알고리즘에 대해 배웁니다.
  - 알고리즘의 성능을 최대화하기 위한 하이퍼파라미터 튜닝을 실습합니다.
  - 여러 트리를 합쳐 일반화 성능을 높일 수 있는 앙상블 모델을 배웁니다.

이 책의 학습 목표

- **CHAPTER 06: 비지도 학습**
  - 타깃이 없는 데이터를 사용하는 비지도 학습과 대표적인 알고리즘을 소개합니다.
  - 대표적인 군집 알고리즘인 k-평균과 DBSCAN을 배웁니다.
  - 대표적인 차원 축소 알고리즘인 주성분 분석(PCA)을 배웁니다.
- **CHAPTER 07: 딥러닝을 시작합니다**
  - 딥러닝의 핵심 알고리즘인 인공 신경망을 배웁니다.
  - 대표적인 인공 신경망 라이브러리인 텐서플로와 케라스를 소개합니다.
  - 인공 신경망 모델의 훈련을 돕는 도구를 익힙니다.
- **CHAPTER 08: 이미지를 위한 인공 신경망**
  - 이미지 분류 문제에 뛰어난 성능을 발휘하는 합성곱 신경망의 개념과 구성 요소에 대해 배웁니다.
  - 케라스 API로 합성곱 신경망을 만들어 패션 MNIST 데이터에서 성능을 평가해 봅니다.
  - 합성곱 층의 필터와 활성화 출력을 시각화하여 합성곱 신경망이 학습한 내용을 고찰해 봅니다.
- **CHAPTER 09: 텍스트를 위한 인공 신경망**
  - 텍스트와 시계열 데이터 같은 순차 데이터에 잘 맞는 순환 신경망의 개념과 구성 요소에 대해 배웁니다.
  - 케라스 API로 기본적인 순환 신경망에서 고급 순환 신경망을 만들어 영화 감상평을 분류하는 작업에 적용해 봅니다.
  - 순환 신경망에서 발생하는 문제점과 이를 극복하기 위한 해결책을 살펴봅니다.

Contents

CHAPTER 04 다양한 분류 알고리즘

CHAPTER 04 다양한 분류 알고리즘

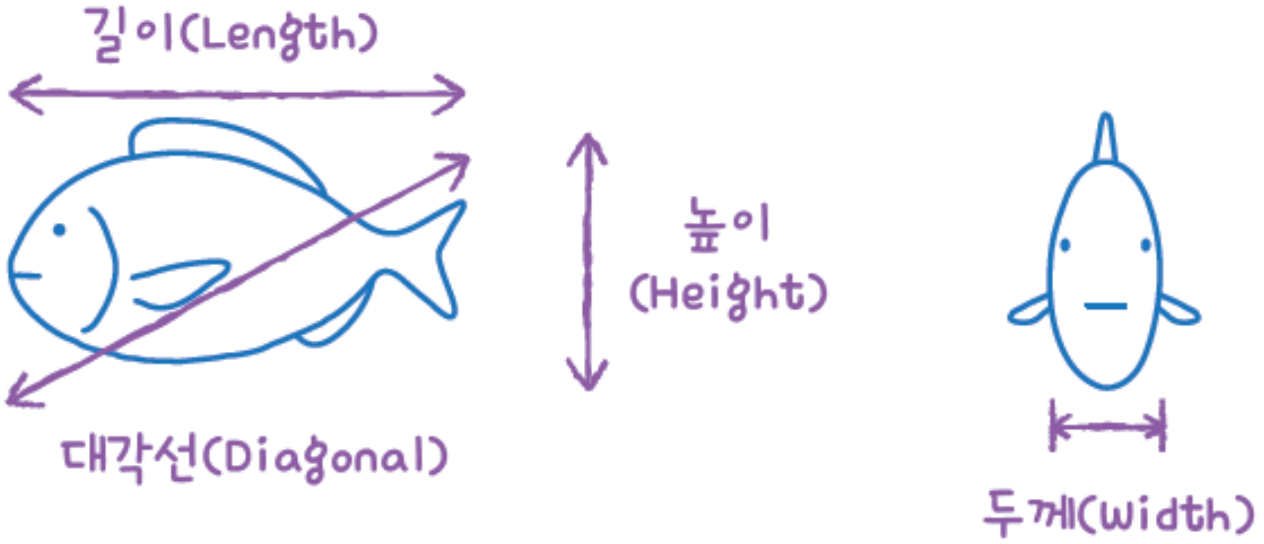
럭키백의 확률을 계산하라!

학습목표

- 로지스틱 회귀, 확률적 경사 하강법과 같은 분류 알고리즘을 배웁니다.
- 이진 분류와 다중 분류의 차이를 이해하고 클래스별 확률을 예측합니다.

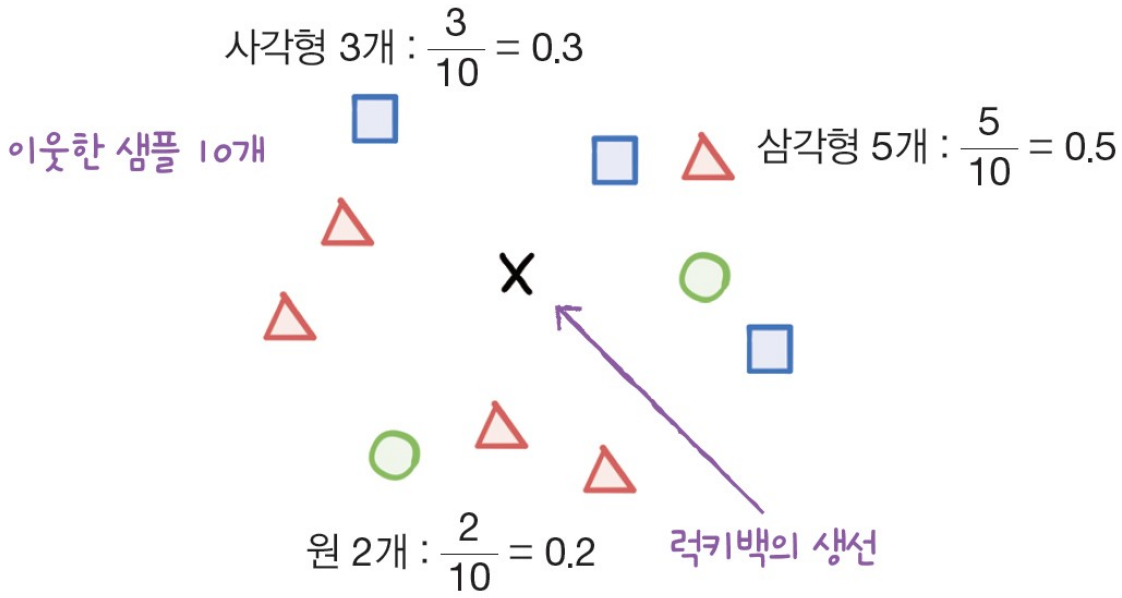
SECTION 4-1 로지스틱 회귀(1)

- **럭키백의 확률**
  - 럭키백에 들어갈 수 있는 생선은 7개
  - 럭키백에 들어간 생선의 크기, 무게 등이 주어졌을 때 7개 생선에 대한 확률을 출력
  - 길이, 높이, 두께 외에도 대각선 길이와 무게도 사용



SECTION 4-1 로지스틱 회귀(2)

- **럭키백의 확률**
  - "k-최근접 이웃은 주변 이웃을 찾아주니까 이웃의 클래스 비율을 확률이라고 출력하면?
    - 샘플 주위에 가장 가까운 이웃 샘플 10개를 표시 - 사각형이 3개, 삼각형이 5개, 원이 2개
    - 이웃한 샘플 X의 클래스를 확률로 삼는다면 샘플 X가 사각형일 확률은 30%, 삼각형일 확률은 50%, 원일 확률은 20%
    - 사이킷런의 k-최근접 이웃 분류기도 이와 동일한 방식으로 클래스 확률을 계산하여 제공



SECTION 4-1 로지스틱 회귀(3)

- **럭키백의 확률**
  - **데이터 준비하기**
    - **판다스로 모델 훈련에 사용할 데이터를 만들기**
    - **판다스의** read_csv( ) **함수로** CSV **파일을 데이터프레임으로 변환한 다음** head( ) **메서드로 처음** 5**개 행을 출력**
      - **소스** https://bit.ly/fish_csv_data

```
import pandas as pd
fish = pd.read_csv('https://bit.ly/fish_csv')
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

SECTION 4-1 로지스틱 회귀(4)

- **럭키백의 확률**
  - **데이터 준비하기**
    - **판다스의** unique( ) **함수를 사용하여, 어떤 종류의 생선이 있는지** Species **열에서 고유한 값을 추출**

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

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

    - **데이터프레임에서** Species **열을 타깃으로 만들고 나머지** 5**개 열은 입력 데이터로 사용**

```
fish_input = fish[['Weight','Length','Diagonal','Height','Width']]
```

    - **데이터프레임에서 여러 열을 선택하면 새로운 데이터프레임이 반환. 이를** fish_input**에 저장**
    - fish_input**에** 5**개의 특성이 잘 저장되었는지 처음** 5**개 행을 출력**

```
fish_input.head( )
```

→

	weight	length	diagonal	height	width
0	242.0	25.4	30.0	11.5200	4.0200
1	290.0	26.3	31.2	12.4800	4.3056
2	340.0	26.5	31.1	12.3778	4.6961
3	363.0	29.0	33.5	12.7300	4.4555
4	430.0	29.0	34.0	12.4440	5.1340

SECTION 4-1 로지스틱 회귀(5)

- **럭키백의 확률**
  - 데이터 준비하기
    - **동일한 방식으로 타깃 데이터 준비**

```
fish_target = fish['Species']
```

    - **데이터를 훈련 세트와 테스트 세트로 나누기**

```
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)
```

    - **사이킷런의** StandardScaler **클래스를 사용해 훈련 세트와 테스트 세트를 표준화 전처리**

      - 훈련 세트의 통계 값으로 테스트 세트를 변환해야 한다는 점에 주의

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

SECTION 4-1 로지스틱 회귀(6)

- **럭키백의 확률**
  - k-**최근접 이웃 분류기의 확률 예측**
    - **사이킷런의** KNeighborsClassifier **클래스 객체를 만들고 훈련 세트로 모델을 훈련한 다음 훈련 세트와 테스트 세트의 점수를 확인**
    - **최근접 이웃 개수인** k**를** 3**으로 지정하여 사용**

```
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
```

    - **타깃 데이터를 만들 때** fish['Species']**를 사용해 만들었기 때문에 훈련 세트와 테스트 세트의 타깃 데이터에도** 7**개의 생선 종류가 들어감**
    - **다중 분류**(multiclass classification): **타깃 데이터에** 2**개 이상의 클래스가 포함**

SECTION 4-1 로지스틱 회귀(7)

- **럭키백의 확률**
  - k-**최근접 이웃 분류기의 확률 예측**
    - **타깃값을 그대로 사이킷런 모델에 전달하면 순서가 자동으로 알파벳 순으로 매겨져** pd.unique(fish['Species'])**로 출력했던 순서와 다르게 됨**
    - KNeighborsClassifier**에서 정렬된 타깃값은** classes_ **속성에 저장됨**

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

      → ['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']
    - predict( ) **메서드 사용, 테스트 세트에 있는 처음** 5**개 샘플의 타깃값을 예측**

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

      → ['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']

SECTION 4-1 로지스틱 회귀(8)

- **럭키백의 확률**
  - k-**최근접 이웃 분류기의 확률 예측**
    - **사이킷런의 분류 모델은** predict_proba( ) **메서드로 클래스별 확률값을 반환**
    - **테스트 세트에 있는 처음** 5**개의 샘플에 대한 확률을 출력**
      - **넘파이** round( ) **함수는 기본으로 소수점 첫째 자리에서 반올림을 하는데,** decimals **매개변수로 유지할 소수점 아래 자릿수를 지정할 수 있음**

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

▲ 소수점 네 번째 자리까지 표기
다섯 번째 자리에서 반올림

→

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

첫 번째 클래스(Bream)에 대한 확률

네 번째 샘플 → [0. 0. 0.6667 0. 0.3333 0. 0.]

두 번째 클래스(Parkki)에 대한 확률

- predict_proba( ) **메서드의 출력 순서는 앞서 보았던** classes_ **속성과 같음 즉 첫 번째 열이** 'Bream'**에 대한 확률,** **두 번째 열이** 'Parkki'**에 대한 확률**

SECTION 4-1 로지스틱 회귀(9)

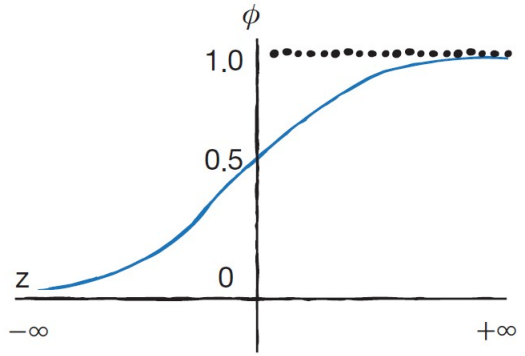
◦ 로지스틱 회귀

- **로지스틱 회귀**(logistic regression)**는 이름은 회귀이지만 분류 모델**
- **이 알고리즘은 선형 회귀와 동일하게 선형 방정식을 학습**

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

- a, b, c, d, e**는 가중치 혹은 계수**
- **특성은 늘어났지만** 3**장에서 다룬 다중 회귀를 위한 선형 방정식과 같음**
- z**는 어떤 값도 가능하지만 확률이 되려면** 0~1(**또는** 0~100%) **사이 값이 되어야 함**
- **시그모이드 함수**(sigmoid function) **또는 로지스틱 함수**(logistic function)**를 사용하여** z**가 아주 큰 음수일 때** 0**이 되고**, z**가 아주 큰 양수일 때** 1**이 되도록 바꿀 수 있음**

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



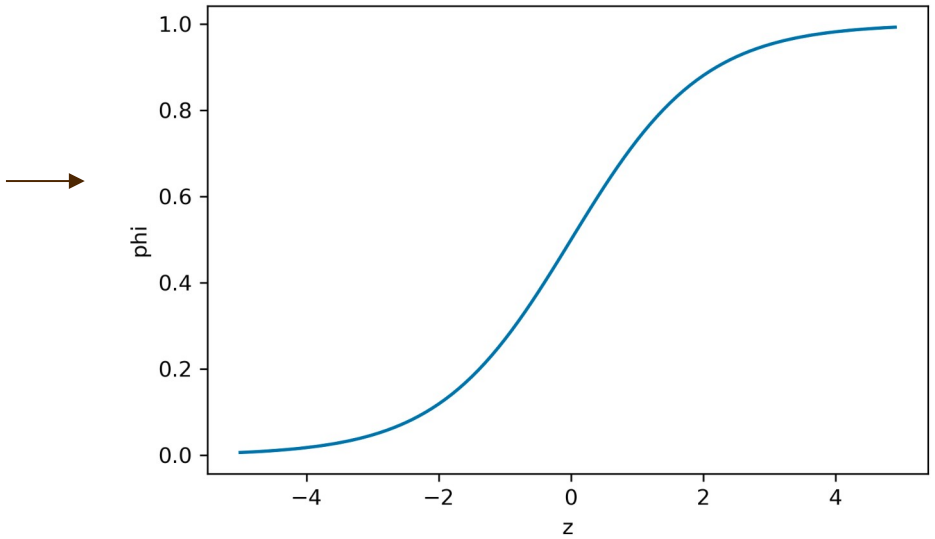
▲ 시그모이드 함수와 그래프

SECTION 4-1 로지스틱 회귀(10)

◦ 로지스틱 회귀

- **넘파이를 사용하여 시그모이드 그래프 작성**
  - **먼저** -5**와** 5 **사이에** 0.1 **간격으로 배열** z**를 만들고**, **그다음** z **위치마다 시그모이드 함수를 계산**
  - **지수 함수 계산은** np.exp( ) **함수를 사용**

```
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()
```



- **이진 분류일 경우 시그모이드 함수의 출력이** 0.5**보다 크면 양성 클래스**, 0.5**보다 작으면 음성 클래스로 판단**
- **정확히** 0.5**일 때는 라이브러리마다 다를 수 있지만**, **사이킷런은** 0.5**일 때 음성 클래스로 판단**

SECTION 4-1 로지스틱 회귀(11)

◦ 로지스틱 회귀

- 로지스틱 회귀로 이진 분류 수행하기

- 불리언 인덱싱(Boolean indexing): 넘파이 배열은 True, False 값을 전달하여 행을 선택 가능
- 다음과 같이 'A'에서 'E'까지 5개의 원소로 이루어진 배열에서 'A'와 'C'만 골라내려면 첫 번째와 세 번째 원소만 True이 고 나머지 원소는 모두 False인 배열을 전달

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

→ ['A' 'C']

- 훈련 세트에서 도미(Bream)와 빙어(Smelt)의 행만 골라내기

  - 도미와 빙어에 대한 비교 결과를 비트 OR 연산자(|)를 사용해 결합

```
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]
```

SECTION 4-1 로지스틱 회귀(12)

◦ 로지스틱 회귀

- 로지스틱 회귀로 이진 분류 수행하기

- **로지스틱 회귀 모델 훈련** - LogisticRegression **클래스는 선형 모델** sklearn.linear_model **패키지 안에 있음**

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

- **훈련한 모델을 사용해** train_bream_smelt**에 있는 처음** 5**개 샘플을 예측**

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

→ ['Bream' 'Smelt' 'Bream' 'Bream' 'Bream']

- predict_proba( ) **메서드로** train_bream_smelt**에서 처음** 5**개 샘플의 예측 확률을 출력**

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

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

▲ 첫 번째 열이 음성 클래스(0)에 대한 확률
두 번째 열이 양성 클래스(1)에 대한 확률

SECTION 4-1 로지스틱 회귀(13)

- **로지스틱 회귀**
  - **로지스틱 회귀로 이진 분류 수행하기**
    - **사이킷런은 타깃값을 알파벳순으로 정렬하여 사용**
    - classes_ **속성에서 확인**

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

→ ['Bream' 'Smelt']

    - **빙어**(Smelt)**가 양성 클래스**. predict_proba( ) **메서드가 반환한 배열 값을 보면 두 번째 샘플만 양성 클래스인 빙어의 확률이 높고, 나머지는 모두 도미**(Bream)**로 예측**
      - [**노트**] **만약 도미**(Bream)**를 양성 클래스로 사용하려면**? 2**장에서 했던 것처럼** Bream**인 타깃값을** 1**로 만들고 나머지 타깃값은** 0**으로 만들어 사용하면 됨**

SECTION 4-1 로지스틱 회귀(14)

- **로지스틱 회귀**
  - **로지스틱 회귀로 이진 분류 수행하기**
    - **로지스틱 회귀가 학습한 계수 확인**

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

→

[[-0.40451732 -0.57582787 -0.66248158 -1.01329614 -0.73123131]] [-2.16172774]

z = -0.405 × (Weight) - 0.576 × (Length) - 0.662 × (Diagonal) - 1.013 × (Height) - 0.731 × (Width) - 2.161

- decision_function ( ) **메서드로** train_bream_smelt**의 처음** 5**개 샘플의** z**값 출력**

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

→

[-6.02991358 3.57043428 -5.26630496 -4.24382314 -6.06135688]

- **파이썬의 사이파이**(scipy) **라이브러리 시그모이드 함수** expit ( )**으로** decisions **배열의 값을 확률로 변환**

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

→

[0.00239993 0.97262675 0.00513614 0.01414953 0.00232581]

SECTION 4-1 로지스틱 회귀(15)

- **로지스틱 회귀**
  - **로지스틱 회귀로 다중 분류 수행하기**
    - LogisticRegression **클래스를 사용해 7개의 생선을 분류해 보면서 이진 분류와 차이점 학습**
    - LogisticRegression **클래스는 기본적으로 반복적인 알고리즘을 사용**
      - max_iter **매개변수에서 반복 횟수를 지정하며 기본값은** 100
      - **여기에 준비한 데이터셋을 사용해 모델을 훈련하면 반복 횟수가 부족하다는 경고가 발생**
      - **충분하게 훈련시키기 위해 반복 횟수를** 1,000**으로 증가시킴**
    - LogisticRegression**은 기본적으로 릿지 회귀와 같이 계수의 제곱을 규제**
      - **이런 규제를** L2 **규제라고 함**
      - **릿지 회귀에서는** alpha **매개변수로 규제의 양을 조절.** alpha**가 커지면 규제도 커짐**
      - LogisticRegression**에서 규제를 제어하는 매개변수는** C. C**는** alpha**와 반대로 작을수록 규제가 커짐** C**의 기본값은** 1
      - **규제를 조금 완화하기 위해** 20**으로 증가시킴**

SECTION 4-1 로지스틱 회귀(16)

- **로지스틱 회귀**
  - **로지스틱 회귀로 다중 분류 수행하기**
    - LogisticRegression **클래스로 다중 분류 모델을 훈련하는 코드**

```
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
```

    - **테스트 세트의 처음 5개 샘플에 대한 예측을 출력**

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

→

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

    - **테스트 세트의 처음 5개 샘플에 대한 예측 확률을 출력(소수점 네 번째 자리에서 반올림)**

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

→

```
[[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]]
```

SECTION 4-1 로지스틱 회귀(17)

◦ 로지스틱 회귀

- 로지스틱 회귀로 다중 분류 수행하기

- 이진 분류는 샘플마다 2개의 확률을 출력하고 다중 분류는 샘플마다 클래스 개수만큼 확률을 출력
  - 여기에서는 7개
- 이 중에서 가장 높은 확률이 예측 클래스가 됨
- 다중 분류일 경우 선형 방정식 확인을 위해, coef_와 intercept_의 크기를 출력

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

→ (7, 5) (7,)

- 5개의 특성을 사용하므로 coef_ 배열의 열은 5개, 행이 7, intercept_도 7개
- 다중 분류는 클래스마다 z 값을 하나씩 계산하고 가장 높은 z 값을 출력하는 클래스가 예측 클래스가 됨
- 이진 분류에서는 시그모이드 함수를 사용해 z를 0과 1 사이의 값으로 변환 - 다중 분류는 이와 달리 소프트맥스(softmax) 함수를 사용하여 7개의 z 값을 확률로 변환

✚ 여기서 잠깐 소프트맥스 함수?

- 시그모이드 함수는 하나의 선형 방정식의 출력값을 0~1 사이로 압축
- 이와 달리 소프트맥스 함수는 여러 개의 선형 방정식의 출력값을 0~1 사이로 압축하고 전체 합이 1이 되도록 함
- 이를 위해 지수 함수를 사용하기 때문에 정규화된 지수 함수라고도 부름

SECTION 4-1 로지스틱 회귀(18)

- **로지스틱 회귀**
  - **로지스틱 회귀로 다중 분류 수행하기**
    - **소프트맥스 함수 계산**

$$e\_sum = ez1 + ez2 + ez3 + ez4 + ez5 + ez6 + ez7$$

- e^{z1}~e^{z7}**을 각각** e_sum**으로 나눔**

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

SECTION 4-1 로지스틱 회귀(19)

- **로지스틱 회귀**
  - **로지스틱 회귀로 다중 분류 수행하기**
    - decision_function ( ) **메서드로** z1~z7**까지의 값을 구한 다음 소프트맥스 함수를 사용해 확률로 변환**
    - **테스트 세트의 처음** 5**개 샘플에 대한** z1~z7**값 계산**

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

→

```
[[ -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 ]]
```

SECTION 4-1 로지스틱 회귀(20)

- **로지스틱 회귀**
  - **로지스틱 회귀로 다중 분류 수행하기**
    - **앞서 구한** decision **배열을** softmax( ) **함수에 전달**
    - softmax( )**의** axis **매개변수는 소프트맥스를 계산할 축을 지정(여기에서는** axis=1**로 지정하여 각 행**, **즉 각 샘플에 대해 소프트맥스를 계산**)
      - **만약** axis **매개변수를 지정하지 않으면 배열 전체에 대해 소프트맥스를 계산**

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

→

```
[[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]]
```

SECTION 4-1 로지스틱 회귀(21)

- 로지스틱 회귀로 확률 예측(문제해결 과정)
  - 문제
    - 럭키백에 담긴 생선이 어떤 생선인지 확률을 예측하고 예측의 근거가 되는 확률을 출력
  - 해결
    - 가장 대표적인 분류 알고리즘 중 하나인 로지스틱 회귀를 사용
    - 로지스틱 회귀는 회귀 모델이 아닌 분류 모델로 선형 회귀처럼 선형 방정식을 사용
    - 하지만 선형 회귀처럼 계산한 값을 그대로 출력하는 것이 아니라 로지스틱 회귀는 이 값을 0~1 사이로 압축(이 값은 마치 0~100% 사이의 확률)
    - 로지스틱 회귀는 이진 분류에서는 하나의 선형 방정식을 훈련. 이 방정식의 출력값을 시그모이드 함수에 통과시켜 0~1 사이의 값을 생성하고, 이 값이 양성 클래스에 대한 확률(음성 클래스의 확률은 1에서 양성 클래스의 확률을 빼면 됨)
    - 다중 분류일 경우에는 클래스 개수만큼 방정식을 훈련하고, 각 방정식의 출력값을 소프트맥스 함수를 통과시켜 전체 클래스에 대한 합이 항상 1이 되도록 만듦
      - 이 값은 각 클래스에 대한 확률로 이해할 수 있음

SECTION 4-1 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - 로지스틱 회귀는 선형 방정식을 사용한 분류 알고리즘
    - 선형 회귀와 달리 시그모이드 함수나 소프트맥스 함수를 사용하여 클래스 확률을 출력
  - 다중 분류는 타깃 클래스가 2개 이상인 분류 문제
    - 로지스틱 회귀는 다중 분류를 위해 소프트맥스 함수를 사용하여 클래스를 예측
  - 시그모이드 함수는 선형 방정식의 출력을 0과 1 사이의 값으로 압축하며 이진 분류를 위해 사용
  - 소프트맥스 함수는 다중 분류에서 여러 선형 방정식의 출력 결과를 정규화하여 합이 1이 되도록 함

SECTION 4-1 마무리(2)

- **핵심 패키지와 함수**
  - scikit-learn
    - LogisticRegression: **선형 분류 알고리즘인 로지스틱 회귀를 위한 클래스**
    - predict_proba( ) **메서드**: **예측 확률을 반환**
    - decision_function( ): **모델이 학습한 선형 방정식의 출력을 반환**

SECTION 4-1 확인 문제

1. 2개보다 많은 클래스가 있는 분류 문제를 무엇이라 부르나?

① 이진 분류 ② 다중 분류 ③ 단변량 회귀 ④ 다변량 회귀

2. 로지스틱 회귀가 이진 분류에서 확률을 출력하기 위해 사용하는 함수는 무엇인가?

① 시그모이드 함수 ② 소프트맥스 함수

③ 로그 함수 ④ 지수 함수

SECTION 4-1 확인 문제

3. decision_function( ) 메서드의 출력이 0일 때 시그모이드 함수의 값은?

① 0 ② 0.25 ③ 0.5 ④ 1

4. 다음 중 LogisticRegression 클래스의 설명으로 올바른 것은 무엇인가요?

① 회귀 문제에 사용하는 모델입니다.

② 매개변수 C의 값을 증가시키면 규제가 강해집니다.

③ decision_function( ) 메서드는 클래스별 확률을 반환합니다.

④ 이진 분류와 다중 분류를 수행할 수 있습니다.

SECTION 4-2 확률적 경사 하강법(1)

- 매주 7개의 생선 중에서 일부를 무작위로 골라 머신러닝 모델을 학습할 수 있게 훈련 데이터를 제공
- 새로운 문제
  - 수산물을 공급하겠다는 곳이 너무 많아 샘플을 골라내는 일이 너무 어려움
  - 추가되는 수산물은 아직 샘플을 가지고 있지도 않음
  - 새로운 생선이 도착하는 대로 가능한 즉시 훈련 데이터를 제공해야 함
  - 어느 생선이 먼저 올지도, 모든 생선이 도착할 때까지 기다릴 수도 없음

SECTION 4-2 확률적 경사 하강법(2)

◦ 점진적인 학습

- **이전에 훈련한 모델을 버리고 다시 새로운 모델을 훈련하는 방식**

  1) **기존의 훈련 데이터에 새로운 데이터를 추가하여 모델을 매일매일 다시 훈련**

     • **시간이 지날수록 데이터가 늘어나, 모델을 훈련하기 위해 서버를 늘려야 함**

  2) **새로운 데이터를 추가할 때 이전 데이터를 버림으로써 훈련 데이터 크기를 일정하게 유지**

     • **데이터를 버릴 때 다른 데이터에 없는 중요한 생선 데이터가 포함되어 있다면 정확한 예측이 불가**

- **점진적 학습 또는 온라인 학습**

  • **대표적인 점진적 학습 알고리즘은 확률적 경사 하강법**(Stochastic Gradient Descent)

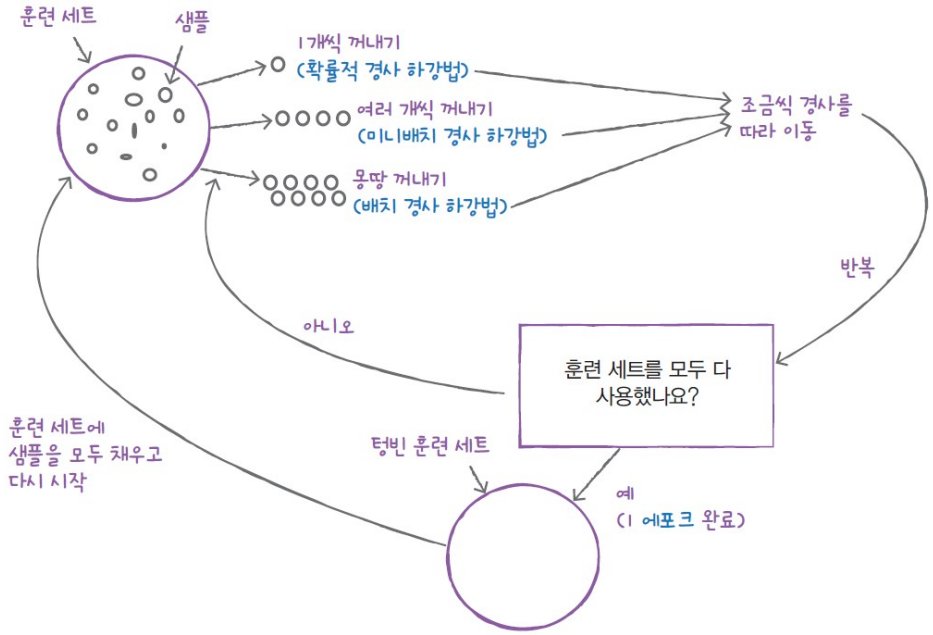
  • **사이킷런에서도 확률적 경사 하강법을 위한 클래스를 제공**

SECTION 4-2 확률적 경사 하강법(3)

- 점진적인 학습
  - **확률적 경사 하강법**
    - **확률적 경사 하강법에서 확률적이란 말은 '무작위하게' 혹은 '랜덤하게'의 기술적인 표현**
    - **'경사'는 기울기를 뜻함**
    - **하강법'은 '내려가는 방법', 즉, 경사 하강법은 경사를 따라 내려가는 방법**
    - **가장 빠른 길은 경사가 가장 가파른 길**
    - **한번에 걸음이 너무 크면 경사를 따라 내려가지 못하고 오히려 올라갈 수도 있음**
    - **전체 샘플을 사용하지 않고 훈련 세트에서 랜덤하게 하나의 샘플을 고르는 것이 확률적 경사 하강법**

SECTION 4-2 확률적 경사 하강법(4)

- 점진적인 학습
  - **확률적 경사 하강법**
    - **에포크**(epoch): **확률적 경사 하강법에서 훈련 세트를 한 번 모두 사용하는 과정**
      - **일반적으로 경사 하강법은 수십, 수백 번 이상 에포크를 수행**
    - **미니배치 경사 하강법**(minibatch gradient descent): **여러 개의 샘플을 사용해 경사 하강법을 수행하는 방식**
    - **배치 경사 하강법**(batch gradient descent): **극단적으로 한 번 경사로를 따라 이동하기 위해 전체 샘플을 사용**



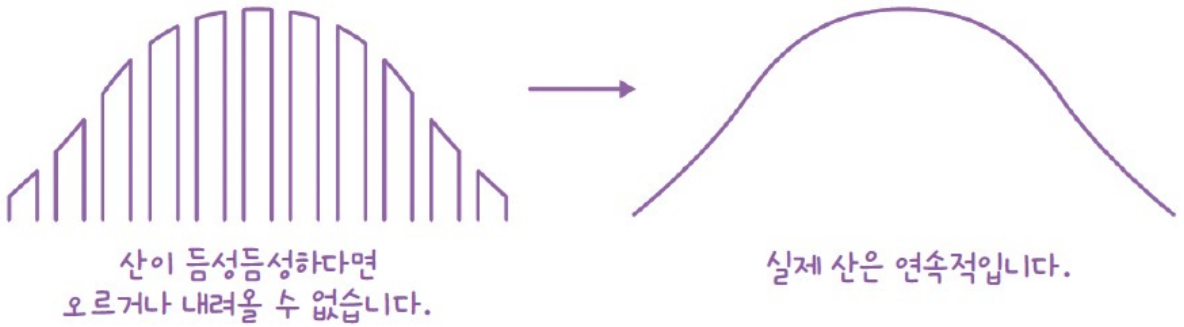
SECTION 4-2 확률적 경사 하강법(5)

◦ 점진적인 학습

- **손실 함수**(loss function)
  - **손실 함수는 어떤 문제에서 머신러닝 알고리즘이 얼마나 엉터리인지를 측정하는 기준**
  - **손실 함수의 값이 작을수록 좋음. 즉, 최솟값 찾기**
  - **생선을 분류하기 위한 손실 함수 사용**
  - **도미와 빙어를 구분하는 이진 분류 문제를 예로 들어 도미는 양성 클래스**(1), **빙어는 음성 클래스**(0)**라고 가정**
  - **연속적인 손실 함수**: **로지스틱 회귀 모델의 확률. 예측은** 0 **또는** 1**이지만 확률은** 0~1 **사이의 어떤 값도 가능**

예측		정답(타깃)
1	=	1
0	≠	1
0	=	0
1	≠	0

▲ 4개의 예측 중에 2개만 맞았으므로 정확도는 1/2 = 0.5



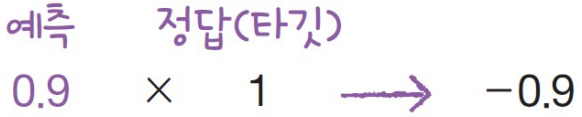
▲ 손실 함수는 미분 가능해야 함

SECTION 4-2 확률적 경사 하강법(6)

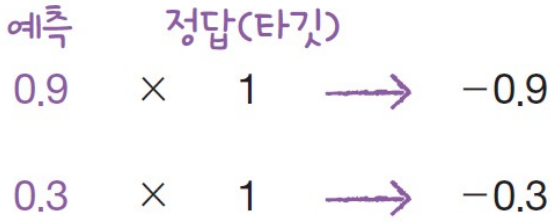
◦ 점진적인 학습

- **로지스틱 손실 함수**

  - **샘플 4개의 예측 확률을 각각 0.9, 0.3, 0.2, 0.8이라고 가정**
  - **첫 번째 샘플의 예측은 0.9이므로 양성 클래스의 타깃인 1과 곱한 다음 음수로 바꿀 수 있음**
    - **이 경우 예측이 1에 가까울수록 좋은 모델 - 예측이 1에 가까울수록 예측과 타깃의 곱의 음수는 점점 작아짐**
    - **이 값을 손실 함수로 사용**



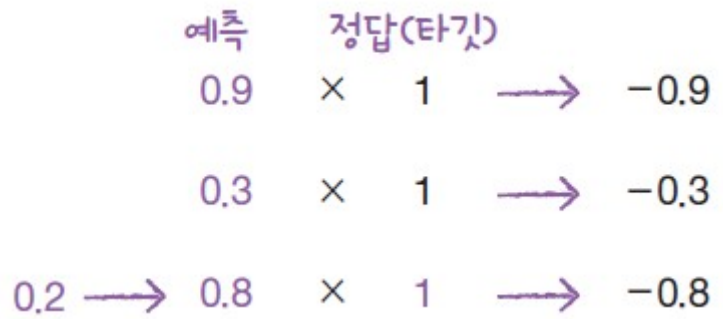
  - **두 번째 샘플의 예측은 0.3**
    - **타깃이 양성 클래스(1)인데 거리가 멀리 있음. 위에서와 마찬가지로 예측과 타깃을 곱해 음수로 변환**
    - **이 값은 -0.3이 되기 때문에 확실히 첫 번째 샘플보다 높은 손실이 됨**



SECTION 4-2 확률적 경사 하강법(7)

◦ 점진적인 학습

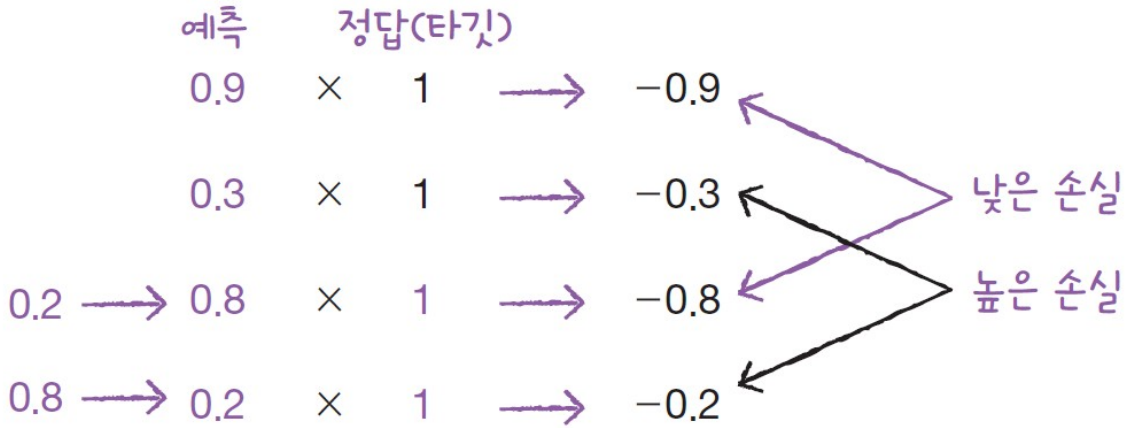
- **로지스틱 손실 함수**
  - **세 번째 샘플의 타깃은 음성 클래스라** 0
    - **이 값을 예측 확률인** 0.2**와 그대로 곱하면 무조건** 0**이 되므로 사용할 수 없음**
    - **타깃을 마치 양성 클래스처럼 바꾸어** 1**로 만들고, 대신 예측값도 양성 클래스에 대한 예측으로 바꿈**
    - **즉,** 1 – 0.2 = 0.8**로 사용. 그다음 곱하고 음수로 바꾸는 것은 위와 동일**



▲ 세 번째 샘플은 음성 클래스인 타깃을 맞추었으므로 손실이 낮아야 함
-0.8은 상당히 낮은 손실임

SECTION 4-2 확률적 경사 하강법(8)

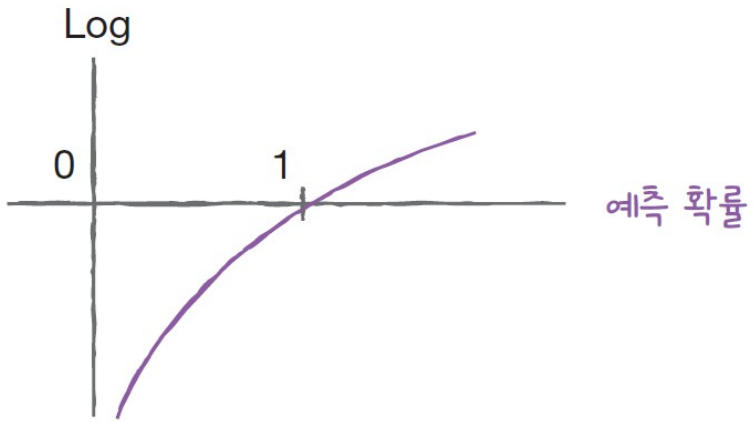
- 점진적인 학습
  - **로지스틱 손실 함수**
    - **네 번째 샘플도 타깃은 음성 클래스**
      - **하지만 정답을 맞히지 못함**
      - **타깃을 1로 바꾸고 예측 확률을 1에서 뺀 다음 곱해서 음수로 바꿈**



▲ 네 번째 샘플의 손실이 높게 나옴
예측 확률을 사용해 이런 방식으로 계산하면 연속적인 손실 함수를 얻을 수 있음

SECTION 4-2 확률적 경사 하강법(9)

- 점진적인 학습
  - **로지스틱 손실 함수**
    - **예측 확률에 로그 함수 적용**
      - **예측 확률의 범위는 0~1 사이인데 로그 함수는 이 사이에서 음수가 되므로 최종 손실 값은 양수**
      - **손실이 양수가 되면 이해하기 더 용이함**
      - **또, 로그 함수는 0에 가까울수록 아주 큰 음수가 되기 때문에 손실을 아주 크게 만들어 모델에 큰 영향을 미칠 수 있음**



타깃 = 1일 때
→ −log(예측 확률)

타깃 = 0일 때
→ −log(1−예측 확률)

- **로지스틱 손실 함수**(logistic loss function)
- **크로스엔트로피 손실 함수**(cross-entropy loss function)

SECTION 4-2 확률적 경사 하강법(10)

- 점진적인 학습
  - **확률적 경사 하강법을 사용한 분류 모델**
  - SGDClassifier
    - fish_csv_data **파일에서 판다스 데이터프레임을 만들기**

```
import pandas as pd
fish = pd.read_csv('https://bit.ly/fish_csv')
```

    - Species **열을 제외한 나머지** 5**개는 입력 데이터로 사용**(Species **열은 타깃 데이터**)

```
fish_input = fish[['Weight','Length','Diagonal','Height','Width']]
fish_target = fish['Species']
```

    - **사이킷런의** train_test_split ( ) **함수를 사용해 이 데이터를 훈련 세트와 테스트 세트로 나누기**

```
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)
```

    - **훈련 세트와 테스트 세트의 특성을 표준화 전처리**

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

SECTION 4-2 확률적 경사 하강법(11)

- 점진적인 학습
  - SGDClassifier
    - SGDClassifier를 sklearn.linear_model **패키지 아래에서 임포트**

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

    - SGDClassifier**의 객체를 만들 때** 2**개의 매개변수를 지정**
      - loss='log'**로 지정하여 로지스틱 손실 함수를 지정**
      - max_iter**는 수행할 에포크 횟수를 지정**. 10**으로 지정하여 전체 훈련 세트를** 10**회 반복**

```
sc = SGDClassifier(loss='log', max_iter=10,
random_state=42)
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```

→

```
0.773109243697479
0.775
```

✚ 여기서 잠깐 ConvergenceWarning 경고가 뜨는데?

- 이 코드를 실행하면 사이킷런은 친절하게도 모델이 충분히 수렴하지 않았다는 ConvergenceWarning 경고를 보냄.
- 이런 경고를 보았다면 max_iter 매개변수의 값을 늘려 주는 것이 바람직
- 오류가 아닌 경고이므로 실습은 이대로 진행해도 무방함

SECTION 4-2 확률적 경사 하강법(12)

- 점진적인 학습
  - SGDClassifier
    - **확률적 경사 하강법은 점진적 학습이 가능**
    - partial_fit( ) **메서드를 사용하여** SGDClassifier **객체를 다시 만들지 않고 훈련한 모델** sc**를 추가로 더 훈련**
      - **이 메서드는** fit( ) **메서드와 사용법이 같지만 호출할 때마다** 1 **에포크씩 이어서 훈련**
      - partial_fit( ) **메서드를 호출하고 다시 훈련 세트와 테스트 세트의 점수를 확인**

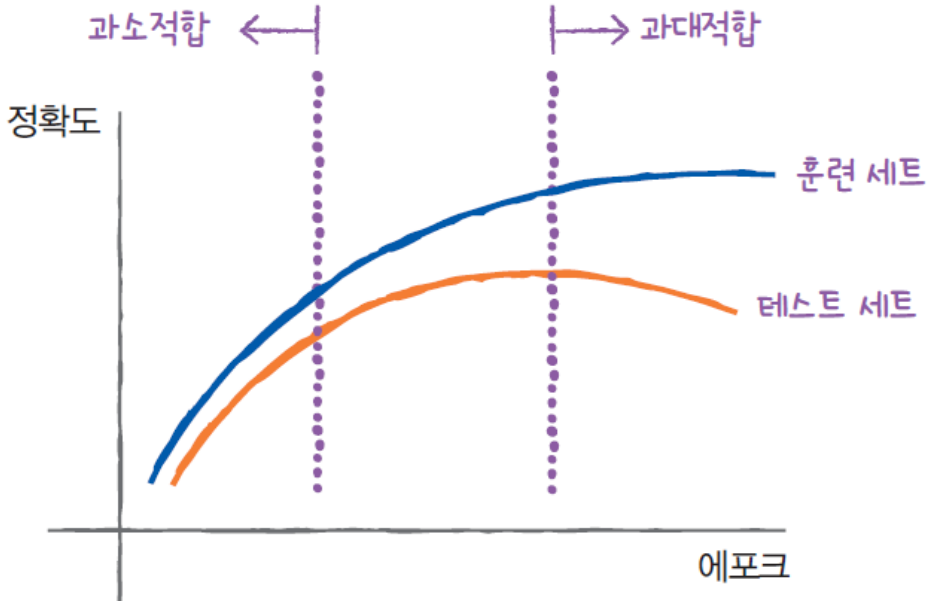
```
sc.partial_fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```

→

```
0.798319327731092
5
0.775
```

SECTION 4-2 확률적 경사 하강법(13)

- 점진적인 학습
  - **에포크와 과대/과소적합**
    - **에포크 횟수가 적으면 모델이 훈련 세트를 덜 학습하고, 횟수가 충분히 많으면 훈련 세트를 완전히 학습**
    - **즉, 적은 에포크 횟수 동안에 훈련한 모델은 훈련 세트와 테스트 세트에 잘 맞지 않는 과소적합된 모델일 가능성이 높고, 반대로 많은 에포크 횟수 동안에 훈련한 모델은 훈련 세트에 너무 잘 맞아 테스트 세트에는 오히려 점수가 나쁜 과대적합된 모델일 가능성이 높음**
    - **조기 종료**(early stopping): **과대적합이 시작하기 전에 훈련을 멈추는 것**



SECTION 4-2 확률적 경사 하강법(14)

◦ 점진적인 학습

- **에포크와 과대/과소적합**
  - **훈련 세트와 테스트 세트의 점수를 그래프로 나타내기**
  - partial_fit( ) **메서드 사용**, **훈련 세트에 있는 전체 클래스의 레이블을** partial_fit( ) **메서드에 전달**
  - **이를 위해** np.unique( ) **함수로** train_target**에 있는** 7**개 생선의 목록을 생성**
  - **또**, **에포크마다 훈련 세트와 테스트 세트에 대한 점수를 기록하기 위해** 2 **개의 리스트를 준비**

```
import numpy as np
sc = SGDClassifier(loss='log', random_state=42)
train_score = []
test_score = []
classes = np.unique(train_target)
```

  - 300**번의 에포크 동안 훈련을 반복하여 진행하고**, **반복마다 훈련 세트와 테스트 세트의 점수를 계산하여** train_score, test_score **리스트에 추가**

```
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))
```

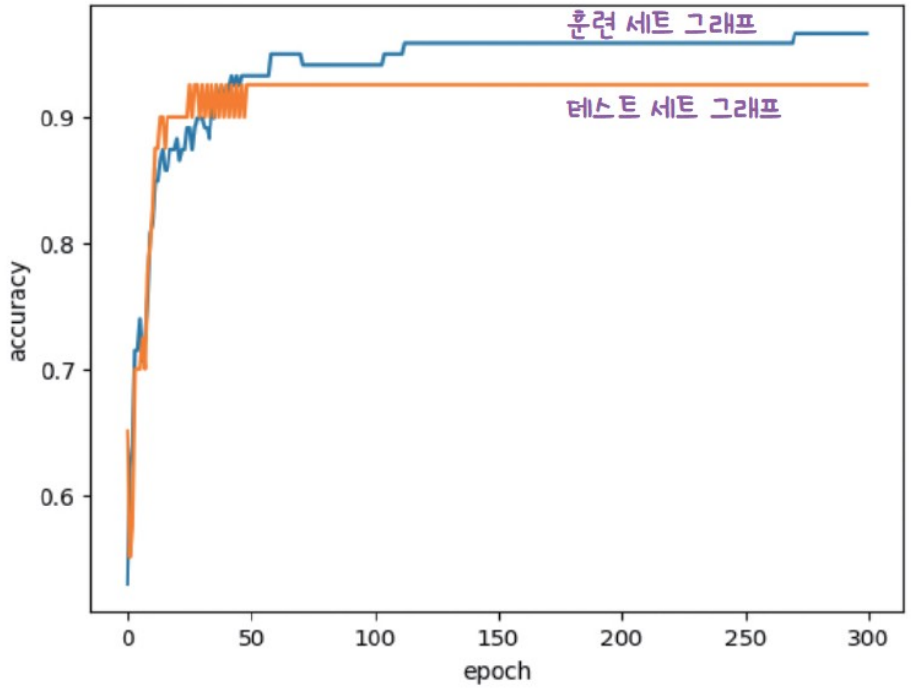
SECTION 4-2 확률적 경사 하강법(14)

◦ 점진적인 학습

- **에포크와 과대/과소적합**

• **300번의 에포크 동안 기록한 훈련 세트와 테스트 세트의 점수 그래프**

```
import matplotlib.pyplot as
plt
plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```



▲ 이 모델의 경우 백 번째 에포크가 적절한 반복 횟수

SECTION 4-2 확률적 경사 하강법(14)

◦ 점진적인 학습

- **에포크와 과대/과소적합**

  - SGDClassifier**의 반복 횟수를** 100**에 맞추고 모델을 다시 훈련하고**, **훈련 세트와 테스트 세트에서 점수를 출력**

```
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
```

  - SGDClassifier**의** loss **매개변수의 기본값은** 'hinge'

    - **힌지 손실**(hinge loss)**은 서포트 벡터 머신**(support vector machine)**이라 불리는 또 다른 머신러닝 알고리즘을 위한 손실 함수**

```
sc = SGDClassifier(loss='hinge', 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))
```

→

```
0.9495798319327731
0.925
```

SECTION 4-2 확률적 경사 하강법(15)

- 점진적 학습을 위한 확률적 경사 하강법(문제해결 과정)
  - 문제
    - 생선을 실시간으로 학습하기 위한 새로운 머신러닝 모델이 필요
  - 해결
    - 확률적 경사 하강법을 사용해 점진적으로 학습하는 로지스틱 회귀 모델을 훈련
    - 확률적 경사 하강법은 손실 함수라는 산을 정의하고 가장 가파른 경사를 따라 조금씩 내려오는 알고리즘
    - 하지만 훈련을 반복할수록 모델이 훈련 세트에 점점 더 잘 맞게 되어 어느 순간 과대적합되고 테스트 세트의 정확도가 줄어듦
    - 요즘엔 대량의 데이터를 이용해 문제를 해결해야 하는 일이 매우 흔한데, 데이터가 매우 크기 때문에 전통적인 머신러닝 방식으로 모델을 만들기 어려움
    - 데이터를 한 번에 모두 컴퓨터 메모리에 읽을 수 없기 때문에, 데이터를 조금씩 사용해 점진적으로 학습하는 방법이 필요
    - 확률적 경사 하강법은 바로 이 문제를 해결하는 핵심 열쇠
      7장에서 신경망을 다룰 때 좀 더 자세히 확률적 경사 하강법을 다시 학습

SECTION 4-2 마무리(1)

◦ 키워드로 끝나는 핵심 포인트

- 확률적 경사 하강법은 훈련 세트에서 샘플 하나씩 꺼내 손실 함수의 경사를 따라 최적의 모델을 찾는 알고리즘
  - 샘플을 하나씩 사용하지 않고 여러 개를 사용하면 미니배치 경사 하강법
  - 한 번에 전체 샘플을 사용하면 배치 경사 하강법
- 손실 함수는 확률적 경사 하강법이 최적화할 대상
  - 대부분의 문제에 잘 맞는 손실 함수가 이미 정의됨
  - 이진 분류에는 로지스틱 회귀(또는 이진 크로스엔트로피) 손실함수를 사용
  - 다중 분류에는 크로스엔트로피 손실 함수를 사용합니다. 회귀 문제에는 평균 제곱 오차 손실 함수를 사용
- 에포크는 확률적 경사 하강법에서 전체 샘플을 모두 사용하는 한 번 반복을 의미
  - 일반적으로 경사 하강법 알고리즘은 수십에서 수백 번의 에포크를 반복

SECTION 4-2 마무리(2)

- **핵심 패키지와 함수**
  - scikit-learn
    - SGDClassifier**는 확률적 경사 하강법을 사용한 분류 모델을 만듦**
      - loss **매개변수는 확률적 경사 하강법으로 최적화할 손실 함수를 지정. 기본값은 서포트 벡터 머신을 위한** 'hinge' **손실 함수. 로지스틱 회귀를 위해서는** l'og'**로 지정**
      - penalty **매개변수에서 규제의 종류를 지정. 기본값은** L2 **규제를 위한** 'l2'. L1 **규제를 적용하려면** 'l1'**로 지정**
      - **규제 강도는** alpha **매개변수에서 지정. 기본값은** 0.0001
      - max_iter **매개변수는 에포크 횟수를 지정. 기본값은** 1000**입니다**.
      - tol **매개변수는 반복을 멈출 조건**. n_iter_no_change **매개변수에서 지정한 에포크 동안 손실이** tol **만큼 줄어들지 않으면 알고리즘이 중단**. tol **매개변수의 기본값은** 0.001, n_iter_no_change **매개변수의 기본값은** 5
    - SGDRegressor**는 확률적 경사 하강법을 사용한 회귀 모델을 만듦**
      - loss **매개변수에서 손실 함수를 지정. 기본값은 제곱 오차를 나타내는** 'squared_loss'
      - **앞의** SGDClassifier**에서 설명한 매개변수는 모두** SGDRegressor**에서 동일하게 사용**

SECTION 4-2 확인 문제

1. **다음 중 표준화 같은 데이터 전처리를 수행하지 않아도 되는 방식으로 구현된 클래스는?**

① KNeighborsClassifier　② LinearRegression

③ Ridge　④ SGDClassifier

2. **경사 하강법 알고리즘의 하나로 훈련 세트에서 몇 개의 샘플을 뽑아서 훈련하는 방식은?**

① 확률적 경사 하강법　② 배치 경사 하강법

③ 미니배치 경사 하강법　④ 부분배치 경사 하강법

SECTION 4-2 확인 문제

3. SGDClassifier 클래스에서 에포크 횟수를 지정하는 매개변수는 무엇인가요?

① max_iter ② epochs

③ shuffle ④ loss

4. 다음 중 경사 하강법에 대해 잘못 설명한 것은 무엇인가요?

① 손실 함수는 샘플 하나에 대한 손실을 정의하고 비용 함수는 모든 샘플에 대한 손실의 합으로 정의됩니다.

② 확률적 경사 하강법은 훈련 샘플을 하나씩 사용해서 손실 함수의 최솟값을 찾습니다.

③ 미니 배치 경사 하강법은 한 번에 여러 개의 샘플을 사용해 손실 함수를 최적화합니다.

④ SGDClassifier는 한 번에 훈련 세트를 모두 사용하는 배치 경사 하강법을 수행합니다.