혼자 공부하는 머신러닝 + 딥러닝

CH03. 회귀 알고리즘과 모델 규제

인공지능공학과 12223557 여예진

목차

1

농어의 길이를 사용해 무게 예측

K-NN 회귀 모델

K-NN 이웃 회귀

= 가장 가까운 이웃 샘플을 찾고 이 샘플들의 타깃값을 평균하여 예측

샘플을 몇 개의 클래스 중 하나로 **분류**하는 문제

임의의 **수치**를 예측하는 문제

K-NN 이웃 회귀

1) 예측하려는 샘플에 가장 가까운 k개의 이웃 선택하기
2) 이웃 샘플의 타깃값들의 **평균** 구하기

1st) 훈련 데이터 준비하기

입력 데이터 + 타깃 데이터 준비하기

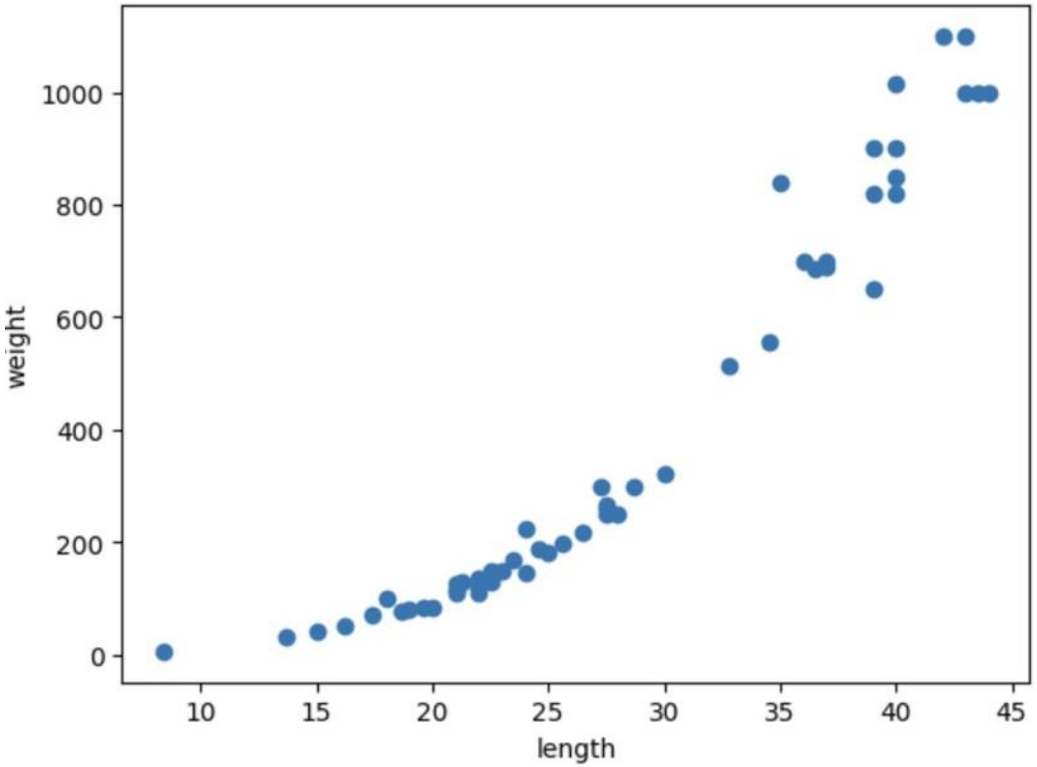
특성 1개

```
# 1. 훈련데이터(입력 + 타깃 데이터) 준비하기
import numpy as np

# 입력데이터 : 농어의 길이 -> 특성
perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0, 21.0,
       21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5, 22.5, 22.7,
       23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3, 27.5, 27.5,
       27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0, 37.0, 37.0,
       39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0, 43.0, 43.5,
       44.0])
# 타깃데이터 : 농어의 무게 -> 타깃
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
       1000.0])
```

1st) 훈련 데이터 준비하기

산점도로 데이터 형태 확인하기

```
# 2. 산점도 그리기 (why? 데이터 형태 확인)
import matplotlib.pyplot as plt

plt.scatter(perch_length, perch_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



농어의 길이 커질수록 농어의 무게도 늘어난다.

2nd) 훈련세트와 테스트세트 나누기

```
# 3. 훈련세트와 테스트 세트 나누기 - 사이킷런 train_test_split()
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(perch_length, perch_weight, random_state=42)
```

```
# 4. 훈련세트(;훈련세트, 테스트세트의 입력데이터) 2차원 배열로 바꾸기 - reshape()
train_input = train_input.reshape(-1, 1)
test_input = test_input.reshape(-1, 1)
print(train_input.shape, test_input.shape)
```

사이킷런에 사용할 특성데이터는 2차원 배열이어야 한다.

3rd) 회귀 모델 훈련하기

KNeighborsRegressor 클래스 이용

```
# 5. 회귀모델 훈련하기 - KNeighborsRegressor 클래스
from sklearn.neighbors import KNeighborsRegressor

# 객체 생성하기
knr = KNeighborsRegressor()
# 객체(모델) 훈련하기 - fit()
knr.fit(train_input, train_target)
```

4th) 회귀 모델 테스트하기

과대적합 or 과소적합

```
# 6. 모델 테스트하기 (테스트세트의 점수 확인)

# 방법1) 결정계수 - score() : 출력값이 높을수록 좋은 것임
print(knr.score(test_input, test_target))

# 방법2) 절댓값 오차 - 타깃과 예측값 사이의 차이 구하기
from sklearn.metrics import mean_absolute_error
# 테스트 세트에 대한 예측
test_prediction = knr.predict(test_input)
# 테스트 세트에 대한 평균 절댓값 오차 계산하기
mae = mean_absolute_error(test_target, test_prediction)
print(mae)
```

방법1) 결정계수(R^2)

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

방법2) 절댓값 오차

```
0.992809406101064
19.157142857142862
```

테스트세트의 점수

```
# 훈련세트의 점수 확인하기
knr.score(train_input, train_target)
# 출력값 : 0.9698823289099254 -> 과소적합
```

훈련세트의 점수 < 테스트세트의 점수 -> 과소적합 문제 발생!!

```
0.9698823289099254
```

훈련세트의 점수

과대적합 & 과소적합

해결책 : 이웃의 개수 k값 조정

과대적합

- **훈련세트 점수 >> 테스트세트 점수**

훈련 세트에만 잘 맞는 모델

과소적합

- **훈련세트 점수 < 테스트세트 점수**
- **두 점수 모두 너무 낮은 경우**
- **훈련세트와 테스트세트의 크기가 매우 작은 경우**

모델이 너무 단순함

5th) 과소적합 문제 해결하기

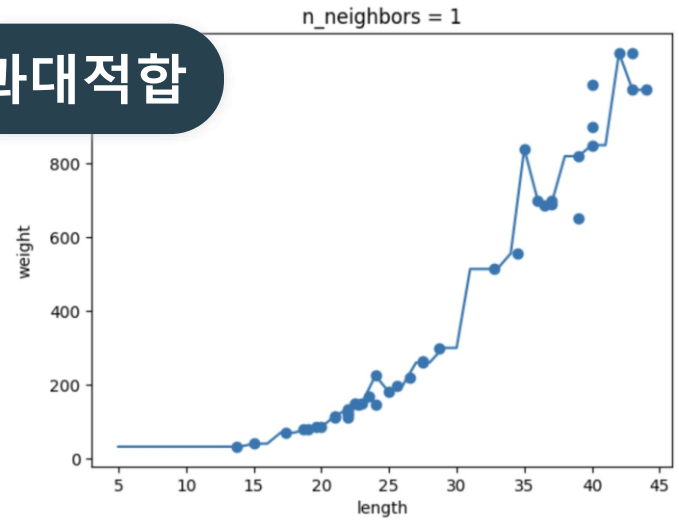
k값 줄여서 모델 복잡하게 만들기

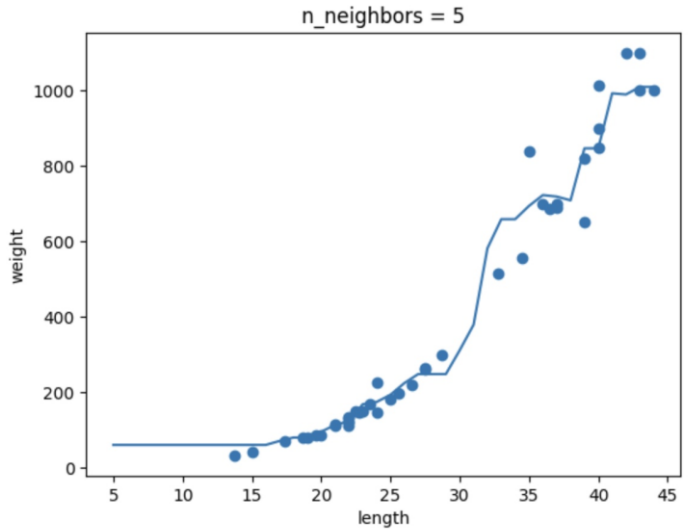
```
# 과소적합 문제 해결하기 -> 모델을 복잡하게 만들기 -> k값 줄이기

# 이웃 개수 3으로 설정하기
knr.n_neighbors = 3
# 모델 다시 훈련시키기
knr.fit(train_input, train_target)

# 훈련세트와 테스트세트 점수 확인하기
print(knr.score(train_input, train_target))
print(knr.score(test_input, test_target))
```

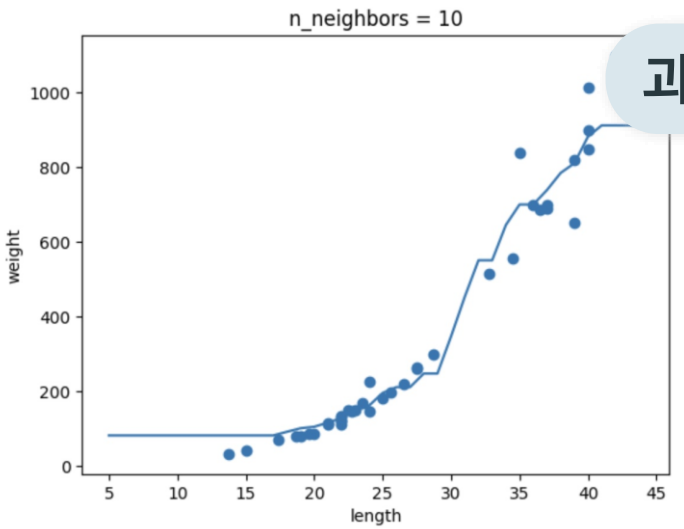
```
0.9804899950518966
0.9746459963987609
```

과소적합 문제 해결) 일반적으로 훈련세트의 점수 > 테스트세트의 점수

과대적합



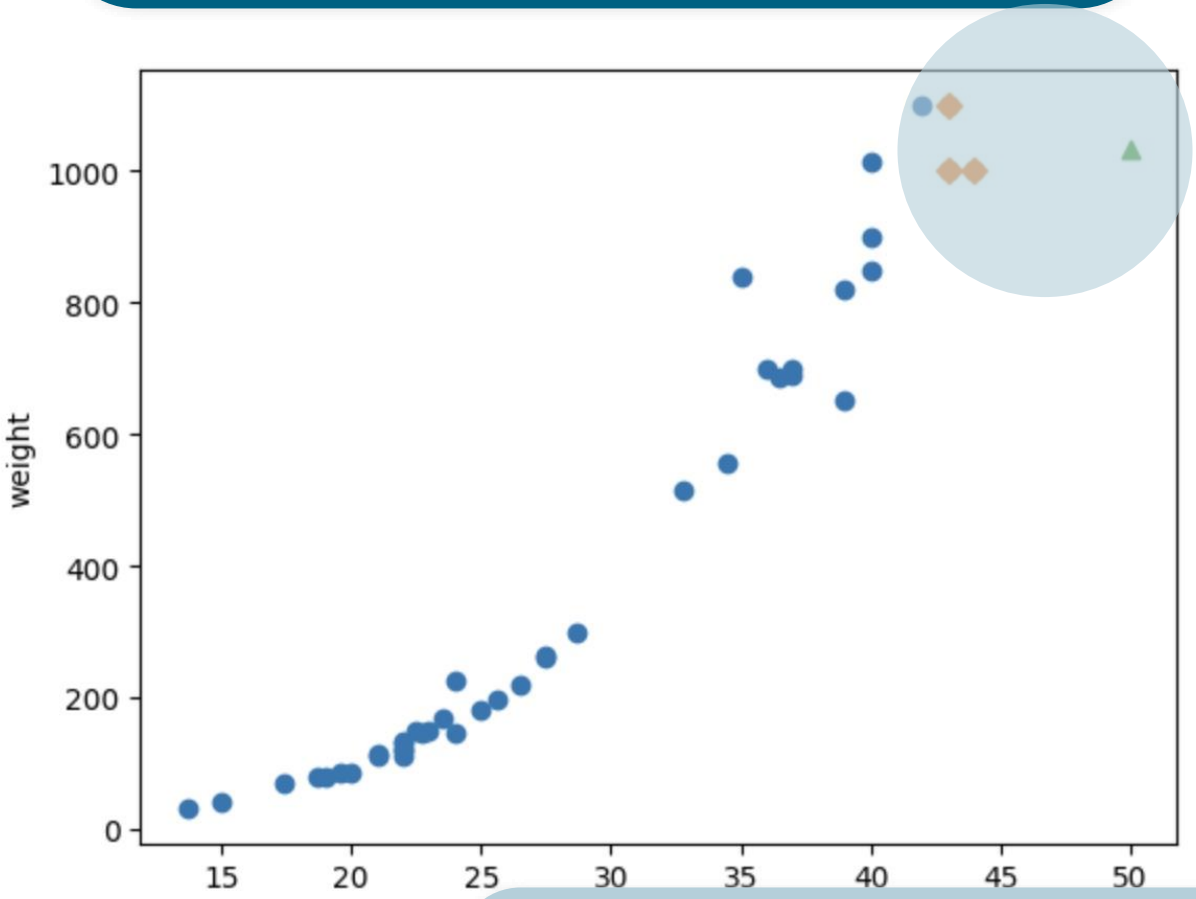




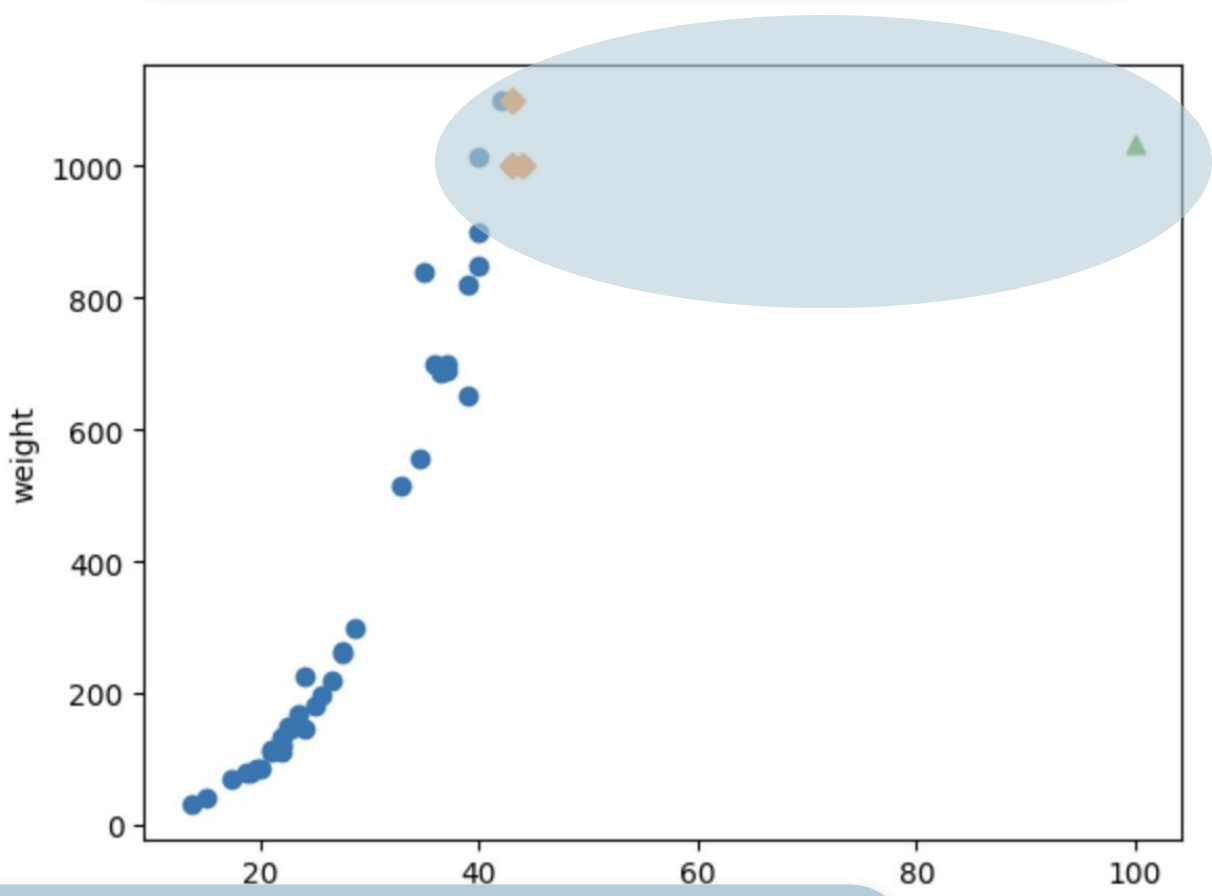
과소적합

Q. 훈련 세트 범위 밖의 샘플도 올바르게 예측할 수 있는가?

50cm 농어의 최근접 이웃



100cm 농어의 최근접 이웃



농어의 길이가 아무리 커도 무게가 더 늘어나지 않는다. -> 문제 발생!!

2

단순 선형 회귀

선형 회귀 모델 1

선형 회귀 (Linear Regression)

= 종속변수 y와 하나 이상의 독립변수 x와의 선형 상관관계를 모델링하는 기법

선형 회귀

단순 선형 회귀

$$y = Wx + b$$

- 적절한 W(가중치)와 b(편향)를 찾는 것
- 그래프의 형태 : 직선

다중 선형 회귀

$$y = W_1x_1 + W_2x_2 + \ldots + W_nx_n + b$$

- 여러 개의 특성을 사용하는 회귀 모델
- 적절한 W와 b를 찾는 것
- 그래프의 형태 : 평면

단순 선형 회귀 모델 훈련하기

LinearRegression 클래스 이용하여 최적의 직선 구하기

```
# LinearRegression의 객체를 만들어 선형회귀 모델 훈련하기

# LinearRegression 클래스의 객체 만들기
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
# 선형회귀 모델 훈련하기
lr.fit(train_input, train_target)
# 50cm 농어에 대해 예측하기
lr.predict([[50]])
```

```
array([1241.83860323])
```

```
# 이 선형회귀가 학습한 직선 구하기 (기울기=계수=가중치, 절편)
print(lr.coef_, lr.intercept_)
```

```
39.01714496] -709.0186449535477
```

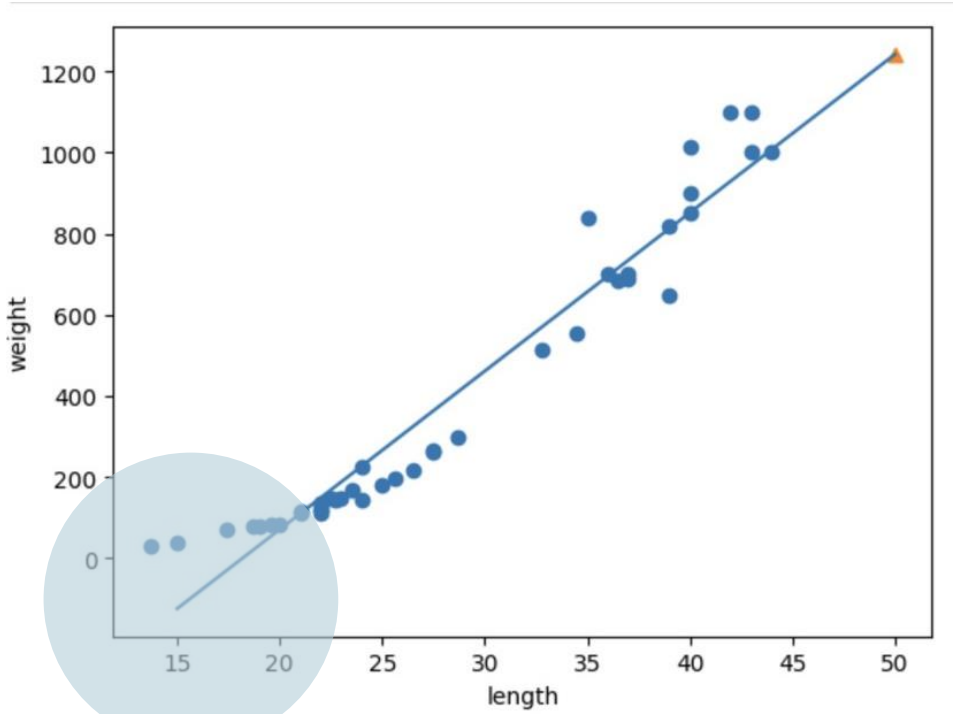
$$y = Wx + b$$

39.01714496 -709.0186449535477

단순 선형 회귀 모델 훈련하기

산점도로 훈련세트와 50cm 농어 샘플 나타내기 / 점수 확인하기

```
# 농어의 길이 15~50까지 직선으로 그리기

# 훈련세트의 산점도 그리기
plt.scatter(train_input, train_target)
# 15~50까지 1차 방정식 그래프 그리기 <- 두 점 잇기
plt.plot([15,50], [15*lr.coef_+lr.intercept_, 50*lr.coef_+lr.intercept_])
# 50cm 농어 데이터
plt.scatter(50, 1241.8, marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



문제1) 그래프 왼쪽 아래로 가면 농어의 무게가 음수가 된다.

```
# 훈련세트와 테스트세트에 대한 결정계수 점수 확인하기
print(lr.score(train_input, train_target))
print(lr.score(test_input, test_target))
# 출력값 : 0.939846333997604, 0.8247503123313558 -> 문제1) 과소적합 (why? 훈련세트, 테스트세트 점수 모두 높지 않음) / 문제2) 그래프 왼쪽 아래 (농(
```

```
).939846333997604
).8247503123313558
```

문제2) 과소적합 : 훈련세트, 테스트세트 점수 모두 높지 않다.

3

다중 선형 회귀

선형 회귀 모델 2

다항 회귀 모델 훈련하기

문제1) 해결하기

```
# 농어의 길이 제곱해서 원래 데이터 앞에 붙이기
train_poly = np.column_stack((train_input**2, train_input))
test_poly = np.column_stack((test_input**2, test_input))
# 새롭게 만든 데이터셋의 크기 확인하기
train_poly.shape, test_poly.shape
```

```
((42, 2), (14, 2))
```

```
# train_poly 사용해 선형 회귀 모델 다시 훈련하기
lr = LinearRegression()
lr.fit(train_poly, train_target)

# 50cm 농어의 무게 예측하기
lr.predict([[50**2, 50]])
```

```
array([1573.98423528])
```

```
# 이 모델이 훈련한 계수와 절편 출력하기
print(lr.coef_, lr.intercept_)
```

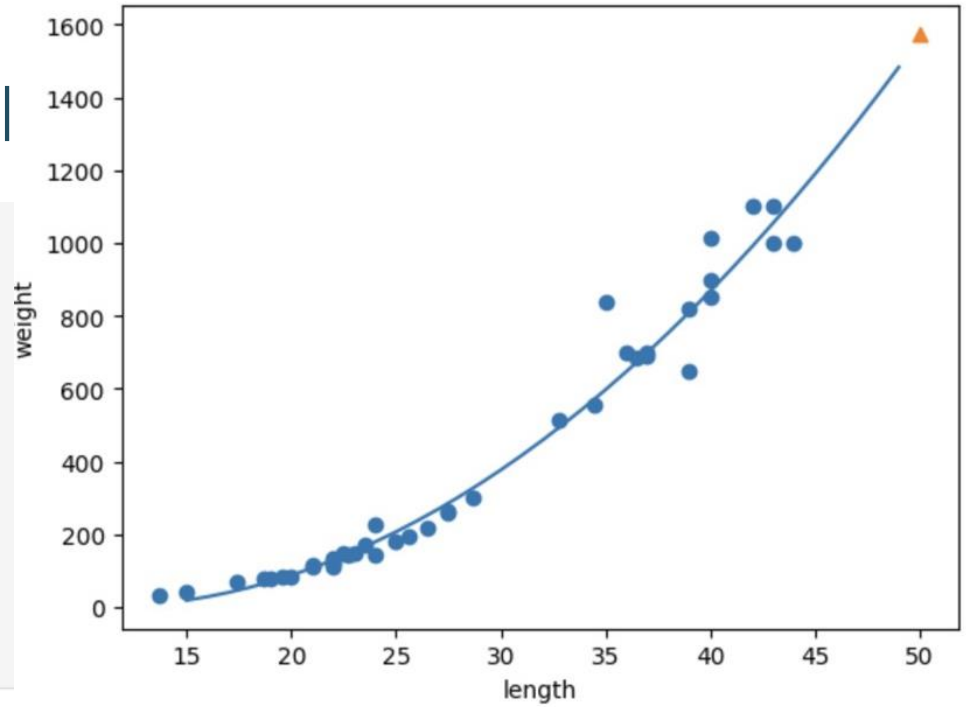
$$y = w_0 + w_1 x + w_2 x^2$$

```
[ 1.01433211 -21.55792498] 116.0502107827827
```

다항 회귀 모델 훈련하기

문제1) 해결하기 -
산점도로 훈련세트와 50cm 농어 샘플 나타내기 / 점수 확인하기

```
# 구간별 직선을 그리기 위해 15-49까지 정수 배열 만들기
point = np.arange(15, 50)
# 훈련 세트의 산점도 그리기
plt.scatter(train_input, train_target)
# 15-49까지 2차 방정식 그래프 그리기
plt.plot(point, 1.01*point**2 - 21.6*point + 116.05)
# 50cm 농어 데이터
plt.scatter(50, 1574, marker='^')

plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```




```
# 훈련세트와 테스트세트에 대한 결정계수 점수 확인하기
print(lr.score(train_poly, train_target))
print(lr.score(test_poly, test_target))
# 출력값 : 0.9706807451768623, 0.9775935108325122 -> 문제) 과소적합이 아직 남아있음
```

```
0.9706807451768623
0.9775935108325122
```

문제2) 과소적합(여전히 해소 안됨) : 테스트세트의 점수가 더 높다.

다중 회귀 모델 훈련하기

문제2) 해결하기 - 데이터 준비하기

```
# 1. 입력데이터 준비하기 - 판다스 이용
import pandas as pd
df = pd.read_csv('https://bit.ly/perch_csv_data')
perch_full = df.to_numpy()
print(perch_full)
```

```
[[ 8.4   2.11  1.41]
 [13.7   3.53  2.  ]
 [15.    3.82  2.43]
 [16.2   4.59  2.63]
 [17.4   4.59  2.94]
 [18.    5.22  3.32]
 [18.7   5.2   3.12]
 [19.    5.64  3.05]
 [19.6   5.14  3.04]
 [20.    5.08  2.77]
 [21.    5.69  3.56]
 [21.    5.92  3.31]
 [21.    5.69  3.67]
 [21.3   6.38  3.53]
 [22.    6.11  3.41]
 [22.    5.64  3.52]
 [22.    6.11  3.52]
 [22.    5.88  3.52]
 [22.    5.52  4.  ]
 [22.5   5.86  3.62]
 [22.5   6.79  3.62]
 [22.7   5.95  3.63]
 [23.    5.22  3.63]
 [23.5   6.28  3.72]
 [24.    7.29  3.72]
 [24.    6.38  3.82]
 [24.6   6.73  4.17]
 [25.    6.44  3.68]
 [25.6   6.56  4.24]
 [26.5   7.17  4.14]
 [27.3   8.32  5.14]
 [27.5   7.17  4.34]
 [27.5   7.05  4.34]
 [27.5   7.28  4.57]
 [28.    7.82  4.2 ]
 [28.7   7.59  4.64]
 [30.    7.62  4.77]
 [32.8  10.03  6.02]
 [34.5  10.26  6.39]
 [35.   11.49  7.8 ]
 [36.5  10.88  6.86]
 [36.   10.61  6.74]
 [37.   10.84  6.26]
 [37.   10.57  6.37]
 [39.   11.14  7.49]
 [39.   11.14  6.  ]
 [39.   12.43  7.35]
 [40.   11.93  7.11]
 [40.   11.73  7.22]
 [40.   12.38  7.46]
 [40.   11.14  6.63]
 [42.   12.8   6.87]
 [43.   11.93  7.28]
 [43.   12.51  7.42]
 [43.5  12.6   8.14]
 [44.   12.49  7.6 ]]
```

```
# 2. 타깃데이터 준비하기
import numpy as np
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
       1000.0])
```

```
# 3. 데이터를 훈련세트와 테스트세트로 나누기
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(perch_full, perch_weight, random_sta
```

다중 회귀 모델 훈련하기

문제2) 해결하기 - 사이킷런의 변환기

변환기(transformer) : 특성을 만들거나 전처리하기 위함
fit() - 새롭게 만들 특성 조합을 찾음
transform() – 실제로 데이터를 변환함

```
# 4. 훈련세트 변환하기 - train_input을 변환한 데이터를 train_poly에 저장하기
poly = PolynomialFeatures(include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)

# train_poly 배열의 크기 확인하기
print(train_poly.shape)

# get_feature_names_out() : 특성의 입력 조합 알려줌
print(poly.get_feature_names_out())
```

PolynomialFeatures 클래스 :
기본적으로 각 특성을 제곱한 항을 추가하고 특성끼리 서로 곱한 항을 추가함

```
(42, 9)
['x0' 'x1' 'x2' 'x0^2' 'x0 x1' 'x0 x2' 'x1^2' 'x1 x2' 'x2^2']
```

특성 : 1개 -> 9개

```
# 5. 테스트세트 변환하기
test_poly = poly.transform(test_input)
```

훈련세트를 기준으로 테스트세트를 변환해야 함

다중 회귀 모델 훈련하기

문제2) 해결하기 - train_poly로 훈련하기

```
# 9개의 특성 - train_poly를 사용해 모델 훈련시키기
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(train_poly, train_target)

# 훈련세트에 대한 점수 확인하기
print(lr.score(train_poly, train_target))
# 테스트세트에 대한 점수 확인하기 -> 점수는 높아지지 않음 but 과소적합 문제 해결
print(lr.score(test_poly, test_target))
```

```
0.9903183436982125
0.9714559911594111
```

과소적합 문제 해결됨

Q. 특성을 더 많이 추가하면 어떻게 될까?

degree 매개변수 : 필요한 고차항의 최대 차수를 지정한다. (기본값 : 2)

```
# degree 매개변수 : 필요한 고차항의 최대 차수 지정 (;특성개수 변경)
poly = PolynomialFeatures(degree=5, include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)
print(train_poly.shape)
```

```
(42, 55)
```

```
# 55개의 특성 - train_poly를 사용해 선형회귀 모델 다시 훈련하기
lr.fit(train_poly, train_target)

# 훈련세트에 대한 점수 확인하기
print(lr.score(train_poly, train_target))
# 테스트세트에 대한 점수 확인하기 -> 아주 큰 음수 나옴 -> 과대적합 -> 문제 발생!! -> 해결1) 특성의 개수 줄이기 / 해결2) 규제
print(lr.score(test_poly, test_target))
```

```
0.9999999999996433
-144.40579436844948
```

테스트세트의 점수가 아주 큰 음수로 나옴 / 과대적합 -> 문제 발생!!

4

릿지 & 라쏘

규제를 추가한 선형 회귀모델

규제 (regularization)

= 머신러닝 모델이 훈련세트에 과대적합되지 않도록 특성에 곱해지는 계수의 크기를 작게 만드는 일

규제

릿지 회귀

- **계수를 제곱한 값**을 기준으로 규제 적용
- 일반적으로 릿지를 더 선호한다.

라쏘 회귀

- **계수의 절댓값**을 기준으로 규제 적용
- 계수의 크기를 0으로 만들 수 있다.

alpha 매개변수 : 규제의 강도 조절

- alpha 값 클수록 -> 규제 강해진다.
- 기본값 : 1

특성의 스케일 정규화하기

규제 적용 전에 실행해야 함

```
# 특성의 스케일 정규화하기 (규제 전)

# StandardScaler 클래스(변환기임) 사용하기
from sklearn.preprocessing import StandardScaler
# 객체 ss 초기화하기
ss = StandardScaler()
# 객체 훈련시키기
ss.fit(train_poly)
# 표준점수로 변환하기
train_scaled = ss.transform(train_poly)
test_scaled = ss.transform(test_poly)
```

릿지 회귀

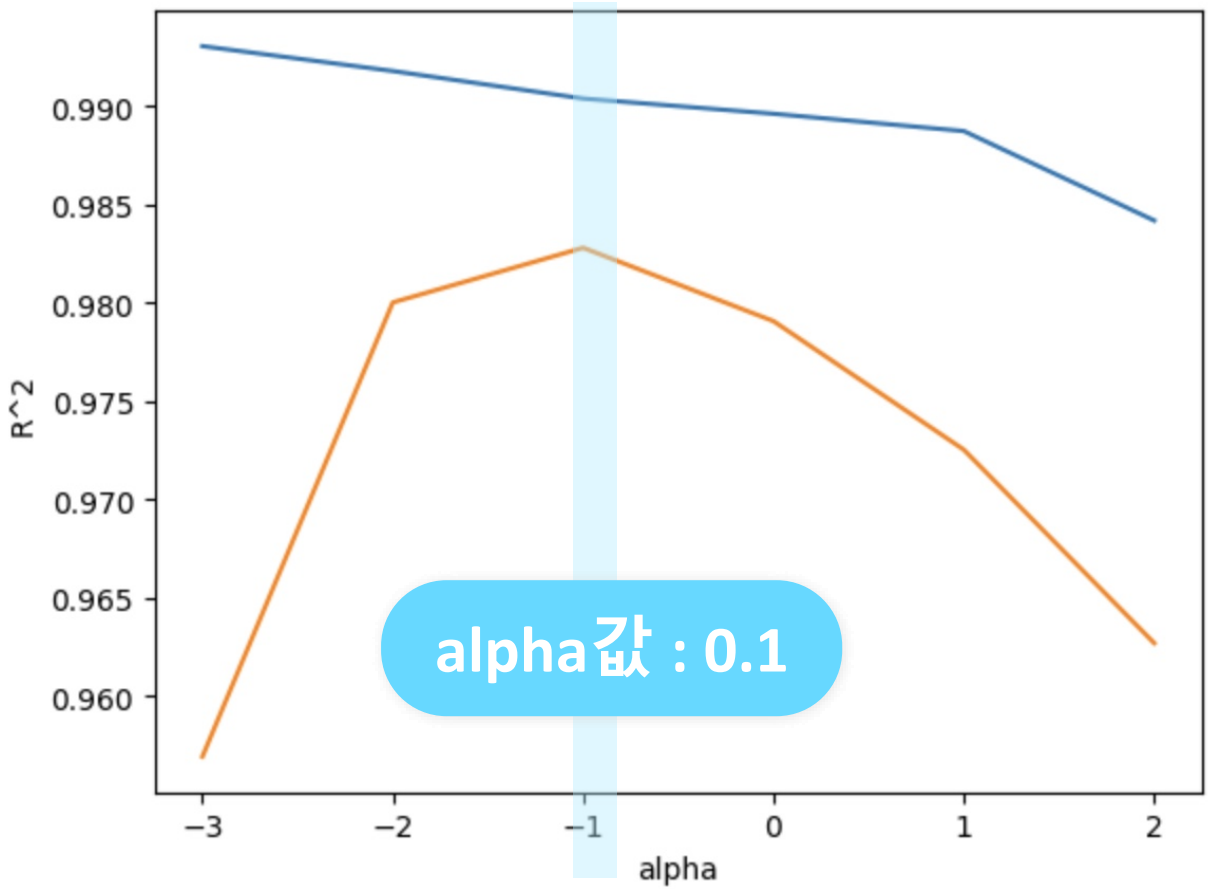
```
# train_scaled 데이터로 릿지 모델 훈련하기
from sklearn.linear_model import Ridge
ridge= Ridge()
ridge.fit(train_scaled, train_target)
print(ridge.score(train_scaled, train_target))
print(ridge.score(test_scaled, test_target))
```

```
0.9896101671037343
0.9790693977615387
```

```
# 적절한 alpha 값 찾는 방법 - 훈련세트와 테스트세트의 점수가 가장 가까운 지점

import matplotlib.pyplot as plt
train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
  # 릿지모델 만들기
  ridge = Ridge(alpha=alpha)
  # 릿지모델 훈련하기
  ridge.fit(train_scaled, train_target)
  # 훈련 점수와 테스트 점수 저장하기
  train_score.append(ridge.score(train_scaled, train_target))
  test_score.append(ridge.score(test_scaled, test_target))

# alpha 값에 대한 결정계수 값의 그래프 그려보기
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.show()
```

```
# alpha값 0.1로 하여 최종 모델 훈련하기
ridge = Ridge(alpha=0.1)
ridge.fit(train_scaled, train_target)
print(ridge.score(train_scaled, train_target))
print(ridge.score(test_scaled, test_target))
```

```
0.9903815817570367
0.9827976465386928
```



라쏘 회귀

```
# 라쏘 모델 훈련하기
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(train_scaled, train_target)
print(lasso.score(train_scaled, train_target))
print(lasso.score(test_scaled, test_target))
```

```
0.989789897208096
0.9800593698421883
```

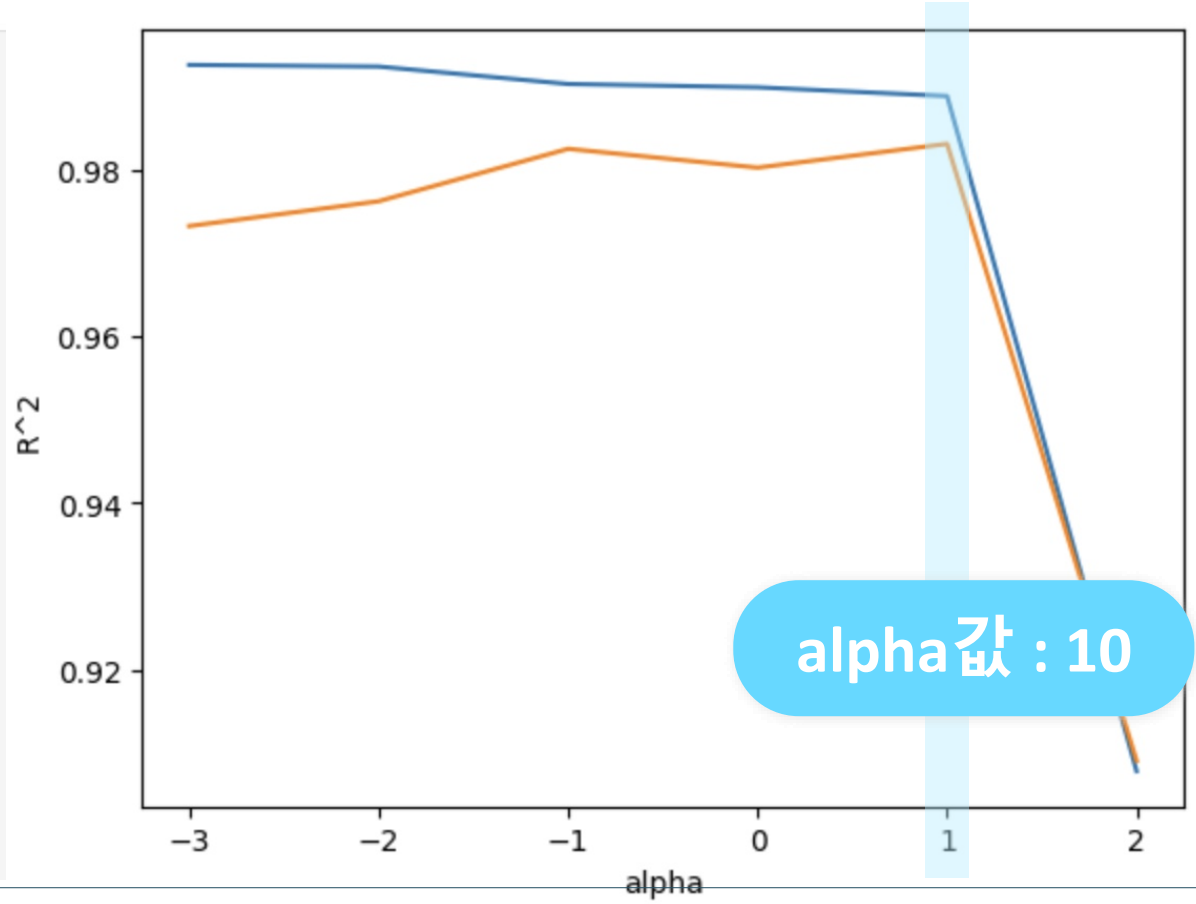
```
# 라쏘 모델의 계수 중 0인 것 찾기
np.sum(lasso.coef_ == 0)
```

```
40
```

```
# alpha값 10으로 하여 최종 모델 훈련하기
lasso = Lasso(alpha=10)
lasso.fit(train_scaled, train_target)
print(lasso.score(train_scaled, train_target))
print(lasso.score(test_scaled, test_target))
```

```
0.9888067471131867
0.9824470598706695
```

```
# 적절한 alpha 값 찾는 방법 - 훈련세트와 테스트세트의 점수가 가장 가까운 지점

train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
  # 라쏘모델 만들기
  lasso = Lasso(alpha=alpha, max_iter=10000)
  # 라쏘모델 훈련하기
  lasso.fit(train_scaled, train_target)
  # 훈련 점수와 테스트 점수 저장하기
  train_score.append(lasso.score(train_scaled, train_target))
  test_score.append(lasso.score(test_scaled, test_target))

# alpha 값에 대한 결정계수 값의 그래프 그려보기
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.show()
```



Thank You