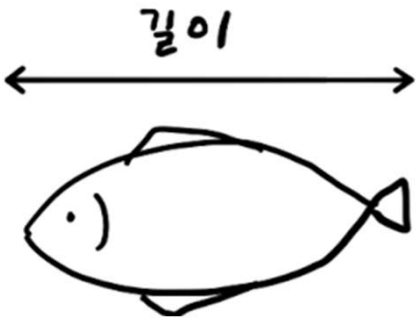
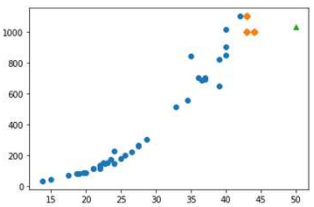
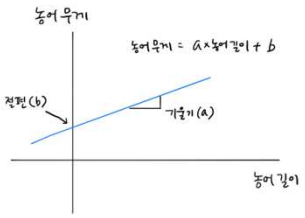
농어의 무게를 예측하라 3
((The Last))



1

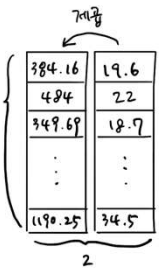
이전까지 내용

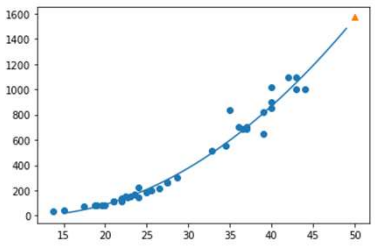




훈련세트보다
테스트 세트의 점수가
여전히 높음!

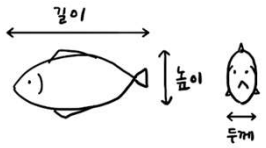
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
# 선형 회귀 모델 훈련
lr.fit(train_input, train_target)

# 50cm 농어에 대한 예측
print(lr.predict([[50]]))
[1241.83860323]

print(lr.coef_, lr.intercept_)
[39.01714496] -709.0186449535477
```



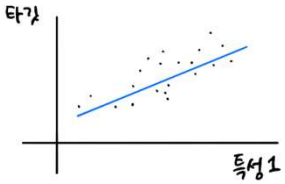


2

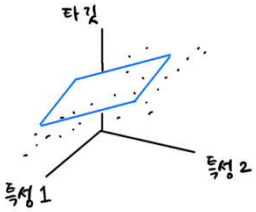
다중 회귀 Multiple regression



여러 개의 특성을 사용한
선형 회귀를 다중 회귀



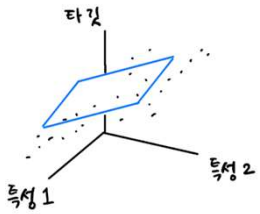
1개의 특성을 사용했을 때
선형 회귀 모델이 학습하는 것은
직선



특성이 2개면 선형회귀는
평면을 학습

3

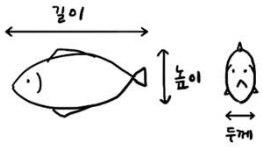
다중 회귀 Multiple regression



- 특성이 2개면 타깃 값과 함께 3차원 공간을 형성
- 선형 회귀 방정식
  *'타깃 = a × 특성1 + b × 특성2 + 절편'*은 평면이 됨
- 그럼 특성이 3개일 경우?
  3차원 공간 그리거나 상상할 수는 없음
- 선형회귀를 단순한 직선이나 평면으로 생각해
  성능이 낮다고 오해 하면 안 됨
- 특성이 많은 고차원에선 선형 회귀가 복잡한 모델을
  표현 가능

4

다중 회귀 Multiple regression



- 이번에는 길이와 함께 높이와 두께도 함께 사용
- 이전과 동일하게 3개의 특성을 각각 제곱해 추가
- 각 특성을 서로 곱해서 또 다른 특성을 만듬
  - Ex) 농어길이 × 농어길이

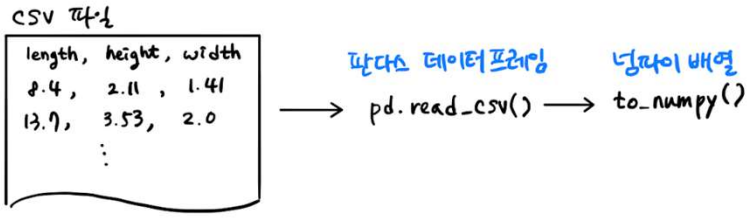
- **특성 공학** Feature engineering
  - 기존의 특성을 사용해 새로운 특성을 뽑아내는 작업
  - 사이킷런에서 해당 기능 제공 : PolynomialFeatures

5

판다스로 데이터 준비

농어의 추가된 특성 3가지가 저장된 csv 파일

```
import pandas as pd

df = pd.read_csv (https://github.com/Gonteer/2024DongALINC/blob/main/001ML/00103_Regression_PolynomialFeatures/perch_full.csv)
perch_full = df.to_numpy()

print(perch_full)
[[ 8.4   2.11  1.41]
 [13.7   3.53  2.  ]
 [15.    3.82  2.43]
 ...
 [43.5  12.6   8.14]
 [44.   12.49  7.6 ]]
```



6

판다스로 데이터 준비

기존 농어 타깃데이터(Weight)

```
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])
```

7

학습/훈련 데이터 준비

```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target =
train_test_split(perch_full, perch_weight,
random_state=42)
```

8

다항 특성 만들기

```
from sklearn.preprocessing import PolynomialFeatures

# degree=2
poly = PolynomialFeatures()  #변환기Transfomer
poly.fit([[2, 3]])   #2개의 특성 2와 3으로 이루어진 샘플 적용 – 새로 만들 특성 조합 찾음

# 1(bias), 2, 3, 2**2, 2*3, 3**2
print(poly.transform([[2, 3]]))  #실제로 데이터를 변환함
```

```
[[1. 2. 3. 4. 6. 9.]]
```

- 2개의 특성(원소)을 가진 샘플 [2,3]이 6개의 특성을 가진 샘플로 바뀜
- PolynomialFeatures 클래스는 기본적으로 각 특성을 제곱한 항을 추가
- 특성끼리 서로 곱한 항을 추가
- 1, 2, 3, 2^2,3^2, 2×3

9

다항 특성 만들기

- 2개의 특성(원소)을 가진 샘플 [2,3]이 6개의 특성을 가진 샘플로 바뀜
- PolynomialFeatures 클래스는 기본적으로 각 특성을 제곱한 항과 특성끼리 서로 곱한 항을 추가
  - 2, 3, 2^2,3^2, 2×3
- 하지만 결과는? `[[1. 2. 3. 4. 6. 9.]]`
- 그럼 왜 1은 추가 되었을까?

10

다항 특성 만들기

$$\textbf{\textit{무게}} = a \times \textit{길이} + b \times \textit{높이} + c \times \textit{두께} + d \times 1$$

- 선형 방정식의 절편은 항상 값이 1인 특성과 곱해지는 계수
- 특성은 (길이, 높이, 두께, 1)로 보임
- 사이킷런의 선형 모델은 자동으로 절편을 추가함
  굳이 이렇게 특성을 만들 필요 없음!!
- PolynomialFeatures()객체 생성할 때 `include=False`로 지정하면 절편을 위한 항이 제거되고, 특성의 제곱과 특성끼리 곱한 항만 추가됨

11

다항 특성 만들기

- PolynomialFeatures()객체 생성할 때 include=False로 지정하면 절편을 위한 항이 제거되고, 특성의 제곱과 특성끼리 곱한 항만 추가됨

```
poly = PolynomialFeatures(include_bias=False)
poly.fit([[2, 3]])
print(poly.transform([[2, 3]]))
```

```
[[2. 3. 4. 6. 9.]]
```

12

LinearRegression(1/2)

```
poly = PolynomialFeatures(include_bias=False)

poly.fit(train_input)
train_poly = poly.transform(train_input)

print(train_poly.shape)
```

```
(42, 9)
```

변환된 데이터의 배열 크기 확인

```
poly.get_feature_names()
```

```
['x0', 'x1', 'x2', 'x0^2', 'x0 x1',
 'x0 x2', 'x1^2', 'x1 x2', 'x2^2']
```

9개의 특성이 각각 어떤 입력의 조합으로 되었는지 확인

```
test_poly = poly.transform(test_input)
```

13

LinearRegression(2/2)

변환된 특성을 사용한 다중 회귀 모델 훈련 → 선형 회기 모델과 훈련 방법 동일

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(train_poly, train_target)

print(lr.score(train_poly, train_target))
0.9903183436982124

print(lr.score(test_poly, test_target))
0.9714559911594134
```

- 높은 점수가 나옴!!
- 농어 길이 뿐만 아니라 높이, 두께 모두 사용
- 각 특성을 제곱하거나 서로 곱해서 다항 특성 더 추가함
- 특성이 늘어나면 선형 회귀의 능력이 강해짐!
- 하지만 *테스트 세트 점수는 높아지지 않았지만 과소적합 문제는 나타나지 않음*

14

더 많은 특성 만들기

```
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)

lr.fit(train_poly, train_target)

print(lr.score(train_poly, train_target))
0.9999999999991097

print(lr.score(test_poly, test_target))
-144.40579242684848
```

- 특성을 더 추가 하면 어떨까?
  3제곱, 4제곱… 항을 넣어봄
- 사이킷런의 PolynomialFeatures 클래스에 degree 매개변수 사용해 필요한 고차항 최대 차수지정
- 여기선 5제곱까지 특성을 만들어 출력 → 특성 개수 55개
- 학습 세트 점수 좋음
- 테스트 세트 점수 ????? 음수!!!

15

더 많은 특성 만들기

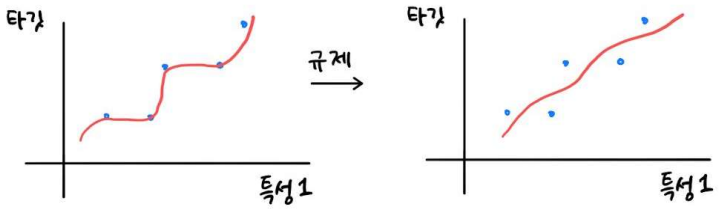
- 특성의 개수를 크게 늘리면 선형 모델은 강력해짐
- 훈련 세트에 대해 거의 완벽하게 학습 할 수 있음
- 이러면 훈련 세트에 너무 *과대적합*이 일어나 테스트 세트에는 형편없는 점수 발생

- 이 문제를 해결하기 위해 특성을 줄여야 함 → 규제 적용

16

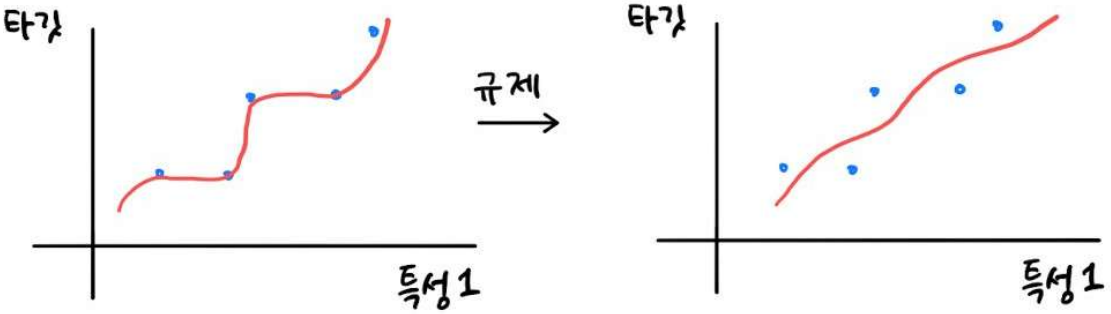
규제 Regularization

- 머신러닝 모델이 훈련 세트를 너무 과도하게 학습하지 못하도록 훼방하는 것을 말함
- 모델이 훈련세트에 과대 적합되지 않도록 만드는 것
- 선형 회귀 모델의 경우 특성에 곱해지는 계수(또는 기울기)의 크기를 작게 만드는 일



17

규제 Regularization



훈련 세트를 과도하게 학습

기울기를 줄여
보다 보편적인 패턴을 학습함

55개의 특성으로 훈련한 선형 회귀 모델의 계수를 규제해
훈련 세트 점수 낮추고, 대신 테스트 세트 점수 높임

18

규제 전에 표준화

- 규제를 적용하기 전 특성의 스케일 정규화 실시
  - 여기에 곱해지는 계수의 값 차이 발생
  - 일반적으로 선형 회귀 모델에 규제를 적용할 때 계수 값의 크기가 서로 많이 다르면 공정하게 제어 안 됨
- 사이킷런의 StandardScaler 클래스 사용 (표준점수 구한 후 변경)

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_poly)

train_scaled = ss.transform(train_poly)
test_scaled = ss.transform(test_poly)
```

19

규제 방법

- 선형 회귀 모델에 규제를 추가한 모델은 릿지와 라쏘라고 부름
- **릿지** Ridge
  - 계수를 제곱한 값을 기준으로 규제 적용
- **라쏘** Lasso
  - 계수의 절대 값을 기준으로 규제 적용
- 일반적으로 **릿지**를 조금 더 선호
- 두 알고리즘 모두 계수의 크기를 줄임
  - 라쏘는 아예 0으로 만들 수 있음
- 사이킷런에서 두 알고리즘 제공

20

릿지 회귀

```
from sklearn.linear_model import Ridge

ridge = Ridge()
ridge.fit(train_scaled, train_target)

print(ridge.score(train_scaled, train_target))
0.9896101671037343

print(ridge.score(test_scaled, test_target))
0.9790693977615386
```

- 테스트 점수 정상으로 돌아옴
- 많은 특성을 사용했음을 감안 했을때 훈련세트가 과대 적합이 되지 않아 테스트 세트에도 좋은 성능 보임

P160

21

적절한 규제 강도 찾기

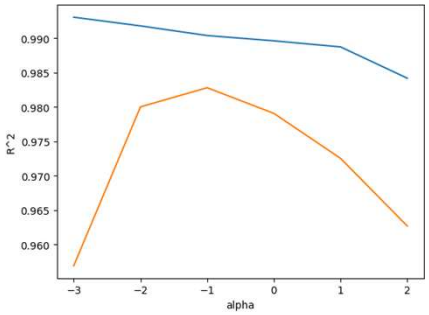
- 릿지와 라쏘 모델을 사용할 때 규제의 양을 임의로 조정 가능
- 모델 객체를 만들 때 alpha 매개변수로 규제의 강도 조절
- Alpha값이 **크면** 규제 강도가 커짐
  → 계수의 값을 더 줄이고 조금 더 과소 적합 되도록 유도
- Alpha 값이 **작으면** 계수를 줄이는 역할이 줄어듬
  → 선형 회귀 모델과 유사해 져 과대적합이 될 가능성 높음

22

랏지 회귀 : 적절한 규제 강도 찾기

적절한 alpha값 찾기 → R^2(결정계수) 값을 구한 후 그래프 그려봄

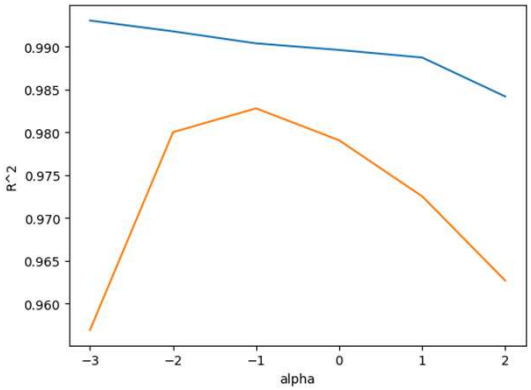
```
import matplotlib.pyplot as plt

train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
    # 릿지 모델을 만듭니다
    ridge = Ridge(alpha=alpha)
    # 릿지 모델을 훈련합니다
    ridge.fit(train_scaled, train_target)
    # 훈련 점수와 테스트 점수를 저장합니다
    train_score.append(ridge.score(train_scaled, train_target))
    test_score.append(ridge.score(test_scaled, test_target))

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()
```



0.001~100까지 10배씩 늘려가며
랏지 회귀 모델 훈련 후
훈련 세트와 테스트 세트의 점수가
가장 가까운 지점이 Alpha값이 됨

23

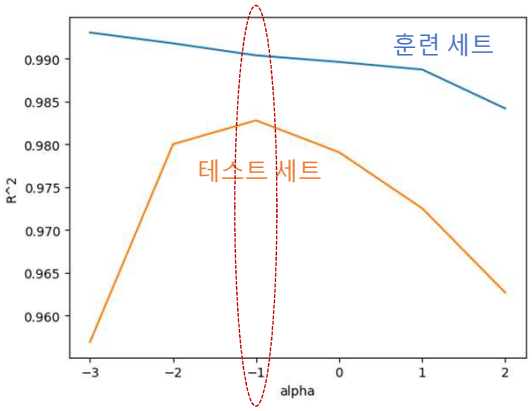
랏지 회귀 : 적절한 규제 강도 찾기



- 그래프는log10() 함수로 지수로 표현해 출력
  - 0.001부터 10배씩 늘렸기 땜에 그래프 왼쪽이 촘촘해 짐
  - alpha_list에 있는 6개의 값을 동일한 간격으로 나태내기 위해 로그 함수로 바꾸어 지수로 표현
  - 0.001 = -3, 0.01= -2, 0.1 = -1 , 1 =0 , 10 =1, 100 =2

24

랏지 회귀 : 적절한 규제 강도 찾기



- 훈련 세트와 테스트 세트의 점수 차이 큼
- [그래프 왼쪽 편]
  훈련 세트에는 좋지만 테스트 세트에서 좋지 않은 과대적합 모습
- [그래프 오른쪽 편]
  훈련 세트와 테스트 세트의 점수가 모두 낮아지는 과소적합의 모습
- 두 그래프가 가깝고 테스트 점수가 가장 높은 것
  → -1 (10^{-1} = 0.1)
- 최종적으로 alpha값을 0.1로 최종 모델 훈련

25

랏지 회귀 : 적절한 규제 강도 찾기

- 최종적으로 alpha값을 0.1로 최종 모델 훈련

```
ridge = Ridge(alpha=0.1)
ridge.fit(train_scaled, train_target)

print(ridge.score(train_scaled, train_target))
0.9903815817570366
print(ridge.score(test_scaled, test_target))
0.9827976465386922
```

- 훈련 세트와 테스트 세트 점수 비슷하게 높고
- 과소/과대 적합 사이에서 균형 이룸

26

라쏘 회귀

```
from sklearn.linear_model import Lasso

lasso = Lasso()
lasso.fit(train_scaled, train_target)

print(lasso.score(train_scaled, train_target))
0.989789897208096

print(lasso.score(test_scaled, test_target))
0.9800593698421883
```

27

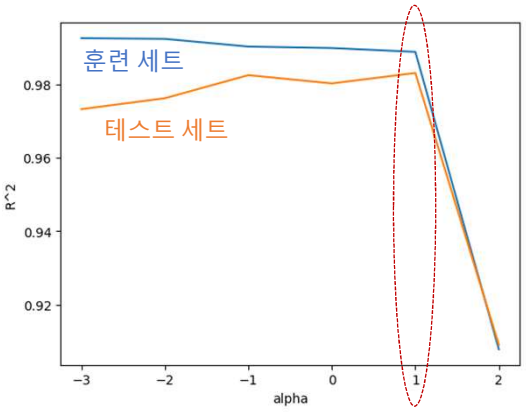
라쏘 회귀 : 적절한 규제 강도 찾기

```
train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
    # 라쏘 모델을 만듭니다
    lasso = Lasso(alpha=alpha, max_iter=10000)
    # 라쏘 모델을 훈련합니다
    lasso.fit(train_scaled, train_target)
    # 훈련 점수와 테스트 점수를 저장합니다
    train_score.append(lasso.score(train_scaled, train_target))
    test_score.append(lasso.score(test_scaled, test_target))
```

28

라쏘 회귀 : 적절한 규제 강도 찾기

```
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()
```



29

라쏘 회귀

```
lasso = Lasso(alpha=10)
lasso.fit(train_scaled, train_target)

print(lasso.score(train_scaled, train_target))
0.9888067471131867
print(lasso.score(test_scaled, test_target))
0.9824470598706695

print(np.sum(lasso.coef_ == 0))
40
```

30

라쏘 회귀

- 라쏘 모델은 계수 값을 아예 0으로 만들 수 있음
- 라쏘 모델 계수는 coef_ 속성에 저장, 이 중에 0인 것(사용 안 된 계수)을 찾아봄

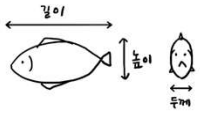
```
print(np.sum(lasso.coef_ == 0))
```

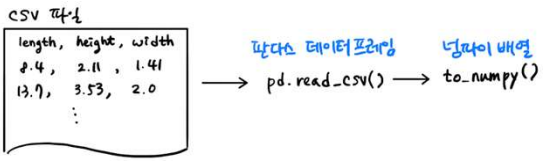
```
40
```

- 위의 결과로 라쏘 모델이 사용한 특성은 15개로 알 수 있음
- 이런 특징 때문에 라쏘 모델의 유용한 특성을 골라내는 용도로 사용함

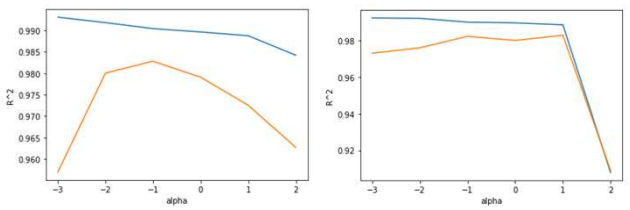
31

지금까지 내용 정리!!






```
poly = PolynomialFeatures(include_bias=False)

poly.fit(train_input)
train_poly = poly.transform(train_input)

print(train_poly.shape)
(42, 9)

poly.get_feature_names()
['x0', 'x1', 'x2', 'x0^2', 'x0 x1',
 'x0 x2', 'x1^2', 'x1 x2', 'x2^2']

test_poly = poly.transform(test_input)
```



32

감사합니다

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

33