머신러닝

- 단순회귀, 다항회귀, 다중회귀
- 특성 공학
- 특성 규제(모델 규제)
- LinearRegression
- PolynomailFeatures
- StandardScaler
- Ridge
- Lasso

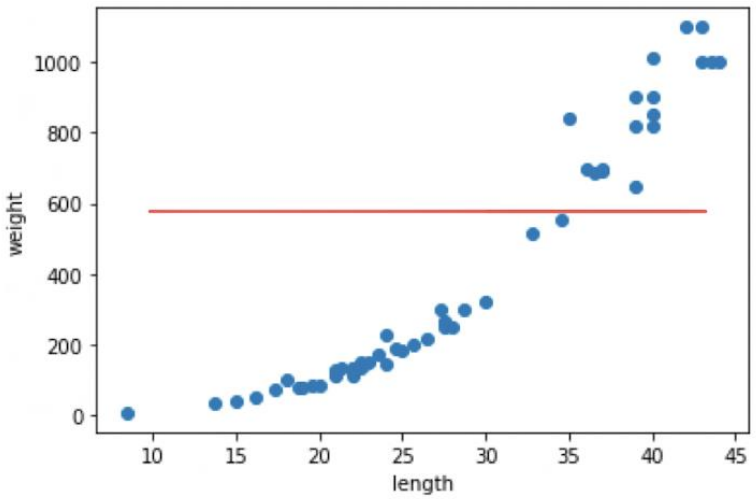
선형 회귀 모델

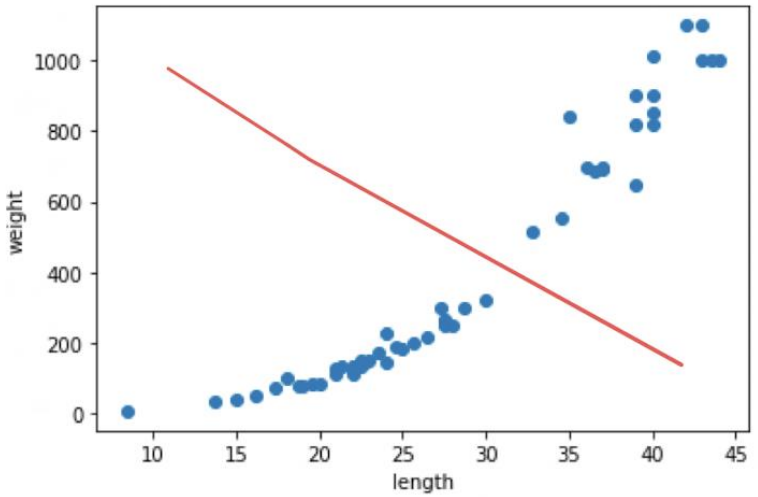
선형회귀모델 개요와 필요성

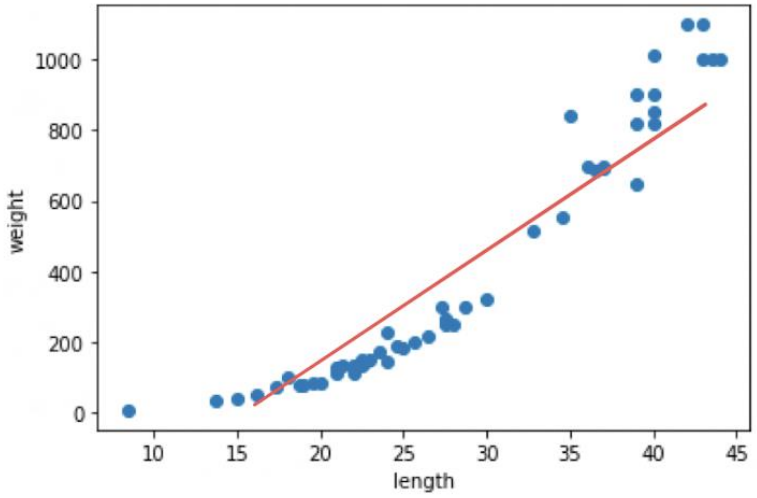
판다스 데이터

• 선형 회귀 모델

어떤 직선이 농어 데이터를 가장 잘 표현할까?







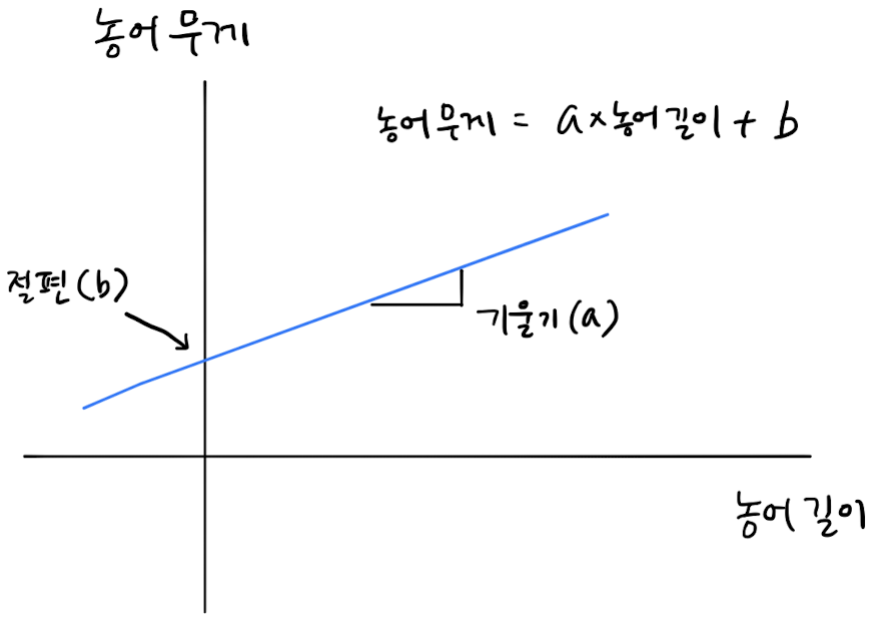
• 사이킷런의 선형회귀 클래스

➢ LinearRegression 클래스 활용한 선형회귀문제 다루기

1. LinearRegression 객체생성
2. fit() 메소드로 모델 학습
3. predict() 메소드로 새로운 샘플데이터(X)의 Y값 예측(추론)
3. score() 메소드로 모델 평가 -> 결정계수 구함
   : 결정계수 값이 클수록 모델의 예측 능력 좋다고 판단함.

• 사이킷런의 선형회귀 클래스

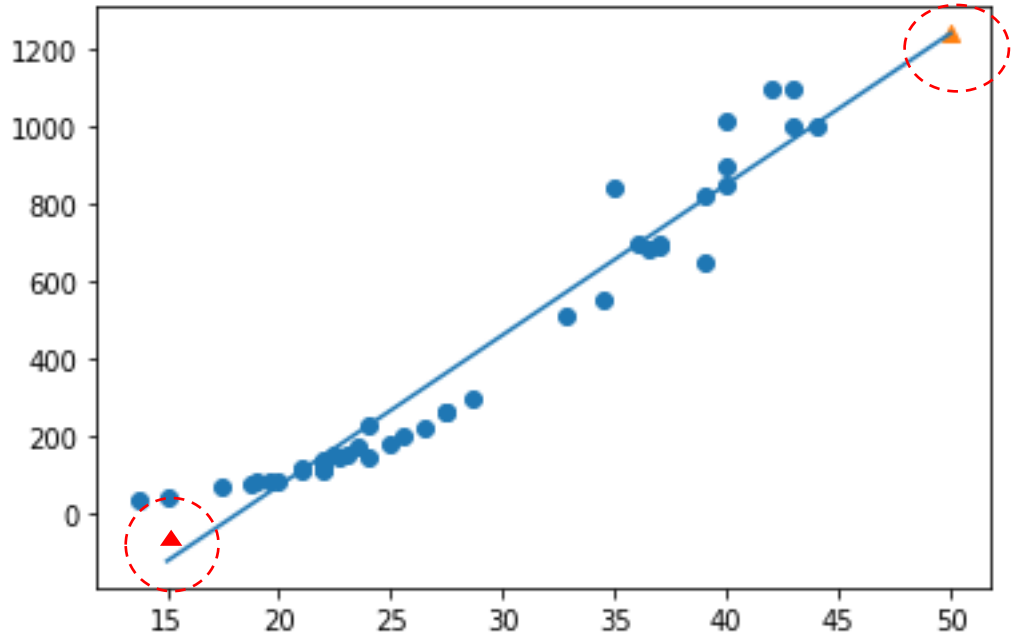
```
from sklearn.linear_model import LinearRegression

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
```



lr.coef:기울기
lr.intercept: 절편

• 사이킷런의 선형회귀 클래스

```
# 훈련 세트의 산점도를 그립니다
plt.scatter(train_input, train_target)

# 15에서 50까지 1차 방정식 그래프를 그립니다
plt.plot([15, 50], [15*lr.coef_+lr.intercept_, 50*lr.coef_+lr.intercept_])

# 50cm 농어 데이터
plt.scatter(50, 1241.8, marker='^')
plt.show()
```




```
print(lr.score(train_input, train_target))
0.9398463339976039

print(lr.score(test_input, test_target))
0.8247503123313558
```

← 결정계수

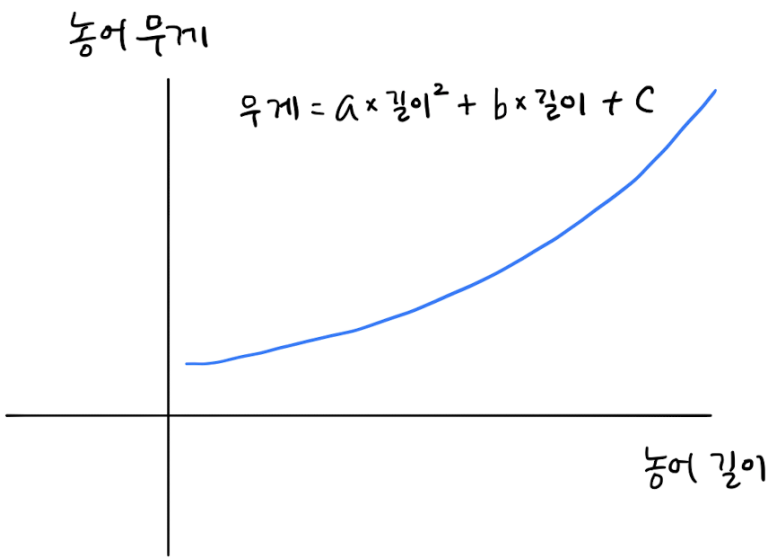
⇒ 결정계수값 산출 결과, 선형회귀모델이 훈련세트에 편향됨을 알 수 있음.
⇒ 단순선형회귀로는 주어진 농어 데이터를 잘 표현한다고 할 수 없음.

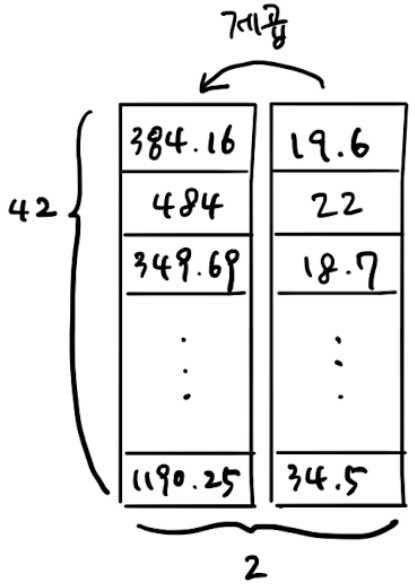
• 다항 선형 회귀(Polynomial Regression)

➢ 주어진 독립 변수만으로 데이터를 잘 표현하는 회귀식이 만들어지지 않을 때, 독립변수로 새로운 특성을 만들어 추가할 수 있음. 새로운 특성을 추가하여 만든 다항회귀모델도 선형회귀에 속함.

예) x를 제곱한 x^2 특성 추가

$y = ax*x + bx + c$





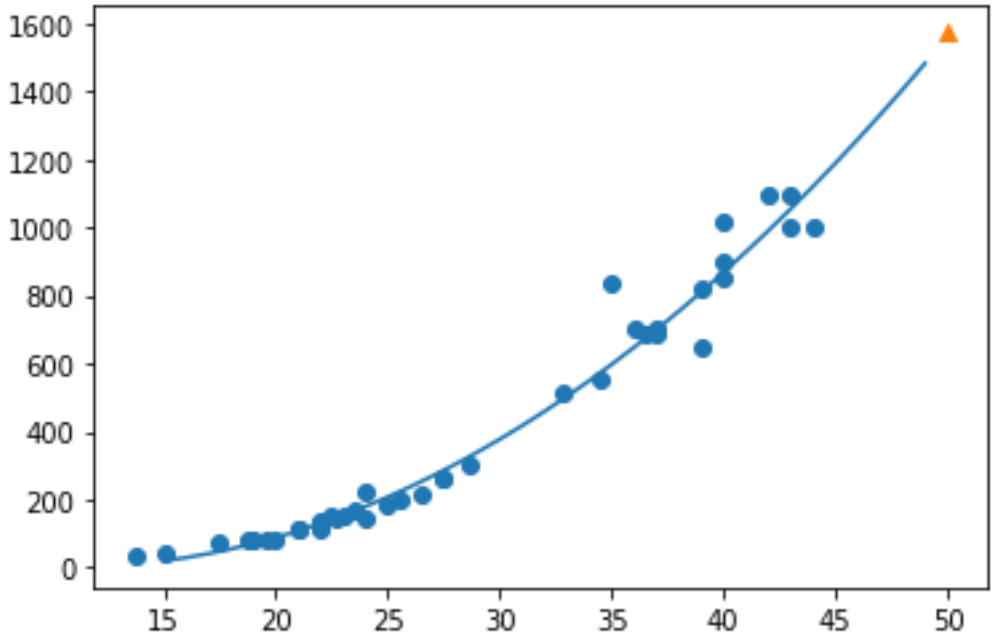
• 다항 선형 회귀(Polynomial Regression)의 예

$$무게 = 1.01 \times 길이^2 - 21.6 \times 길이 + 116.05$$

```
train_poly = np.column_stack((train_input ** 2, train_input))
test_poly = np.column_stack((test_input ** 2, test_input))


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

print(lr.predict([[50**2, 50]]))
[1573.98423528]

print(lr.coef_, lr.intercept_)
[  1.01433211 -21.55792498] 116.0502107827827
```

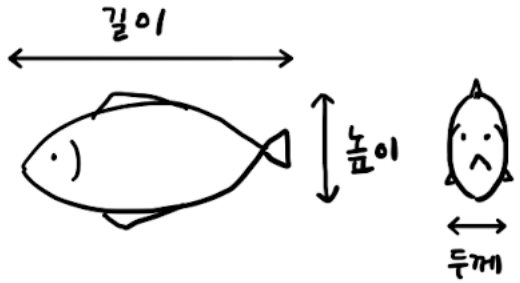
• 다항 선형 회귀(Polynomial Regression)의 예

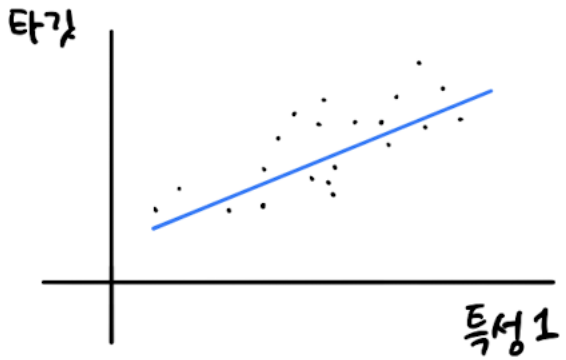
```
# 구간별 직선을 그리기 위해 15에서 49까지 정수 배열을 만듭니다
point = np.arange(15, 50)

# 훈련 세트의 산점도를 그립니다
plt.scatter(train_input, train_target)

# 15에서 49까지 2차 방정식 그래프를 그립니다
plt.plot(point, 1.01*point**2 - 21.6*point + 116.05)

# 50cm 농어 데이터
plt.scatter([50], [1574], marker='^')
plt.show()

print(lr.score(train_poly, train_target))
0.9706807451768623
print(lr.score(test_poly, test_target))
0.9775935108325122
```

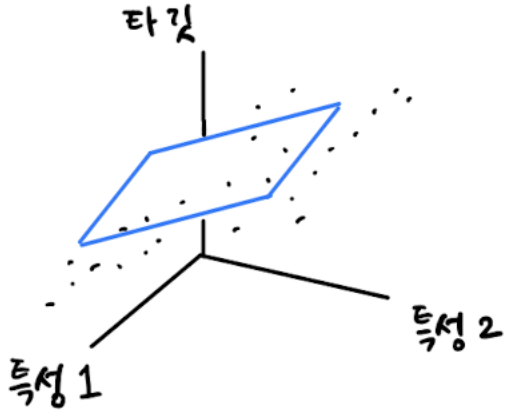


⇒ 선형회귀분석은 훈련 세트 범위를 벗어나는 새로운 샘플에 대한 값 예측 가능함
⇒ 단순선형회귀모델보다 특성 추가한 다항선형회귀모델이 향상되었음

• 다중 선형 회귀(Multiple Linear Regression)

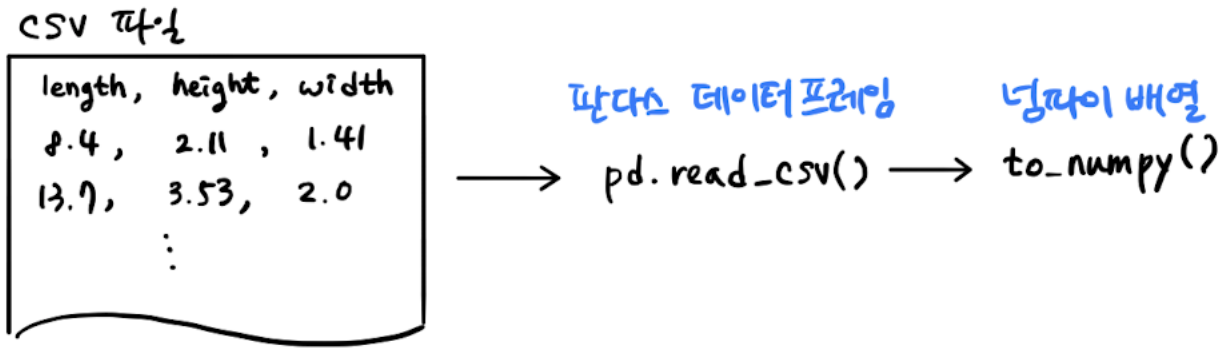






• 다중 선형 회귀(Multiple Linear Regression)

➢ 판다스로 데이터 준비




```
import pandas as pd

df = pd.read_csv('https://bit.ly/perch_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 ]]
```

• 다중선형회귀모델 성능 향상 기법

➢ 다항 특성 만들기

➢ 사이킷런의 특성 생성을 위한 클래스

- PolynomialFeatures 클래스

• fit(): 주어진 샘플데이터로부터 특성조합을 찾는 함수

• transform(): 새로 만들어진 특성에 맞게 샘플데이터 변환하는 함수

• 특성공학(feature engineering)

- 기존 특성을 이용하여 다항 특성 만드는 예

```
from sklearn.preprocessing import PolynomialFeatures

# degree=2
poly = PolynomialFeatures()
poly.fit([[2, 3]])
```

← 특성 2, 3으로 이루어진 샘플로 부터 특성 조합을 찾음

```
# 1(bias), 2, 3, 2**2, 2*3, 3**2
print(poly.transform([[2, 3]]))
[[1. 2. 3. 4. 6. 9.]]
```

← 특성 2, 3으로 이루어진 샘플에 특성조합 적용하여 샘플을 생성

• 다중 선형 회귀(Multiple Linear Regression)

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

test_poly = poly.transform(test_input)
```

```
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
```

새로운 특성을 추가하여 선형회귀 모델생성하고 성능 테스트한 결과
⇒ 테스트데이터의 점수가 훈련데이터의 예측 점수보다 높지 않음
⇒ 과소 적합 문제 해결

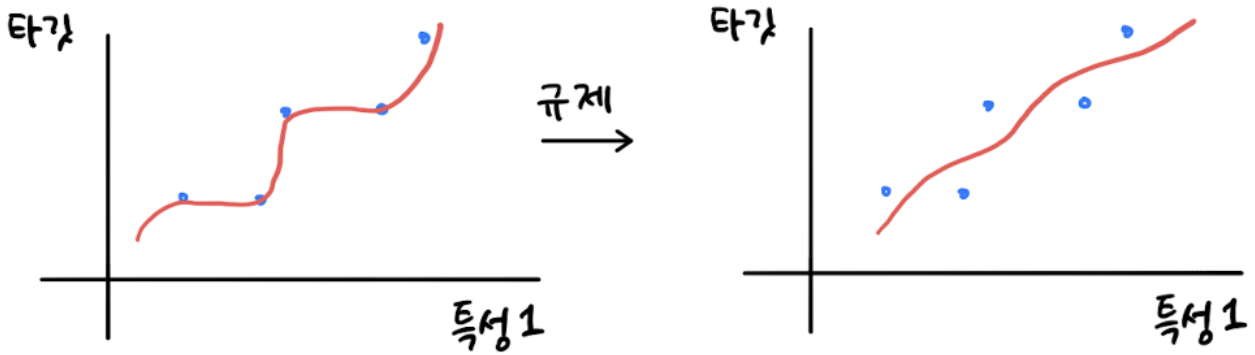
• 다중 선형 회귀(Multiple Linear Regression)

➢ 더 많은 특성으로 선형회귀모델을 만든다면 모델 성능은 좋아질 것인가?

```
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)
```

← 만들어진 특성 개수가 55개

```
lr.fit(train_poly, train_target)

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

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



** 특성이 지나치게 많으면 과대적합 발생함

• 다중 선형 회귀(Multiple Linear Regression)

➢ 사이킷런에서 제공하는 데이터 전처리(표준화) 클래스

k-NN 알고리즘에서 빙어와 도미의 길이와 무게 특성 기준을 맞추기 위해 평균, 표준편차에 의한 특성값 스케일링 내용과 동일한 결과 가져옴

- StandardScaler 클래스
  - fit(), transform()

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_poly)

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

• 선형회귀모델 정규화(규제)

➢ 선형회귀모델의 과적합 방지 -> 모델의 일반화 성능을 향상

Ridge 회귀 (L2 정규화)

• **정의**: Ridge 회귀는 선형회귀의 손실 함수에 **가중치의 제곱합**을 더하여 과적합을 방지

$$\text{손실 함수} = \text{오차 제곱합} + \lambda \sum_i w_i^2$$

λ: 정규화 강도를 조절하는 하이퍼파라미터

Lasso 회귀 (L1 정규화)

•**정의**: Lasso 회귀는 선형회귀의 손실 함수에 **가중치의 절댓값의 합**을 더하여 과적합을 방지

$$\text{손실 함수} = \text{오차 제곱합} + \lambda \sum_i |w_i|$$

λ: 정규화 강도를 조절하는 하이퍼파라미터

• 선형회귀모델 규제

➢ 릿지 회귀 모델

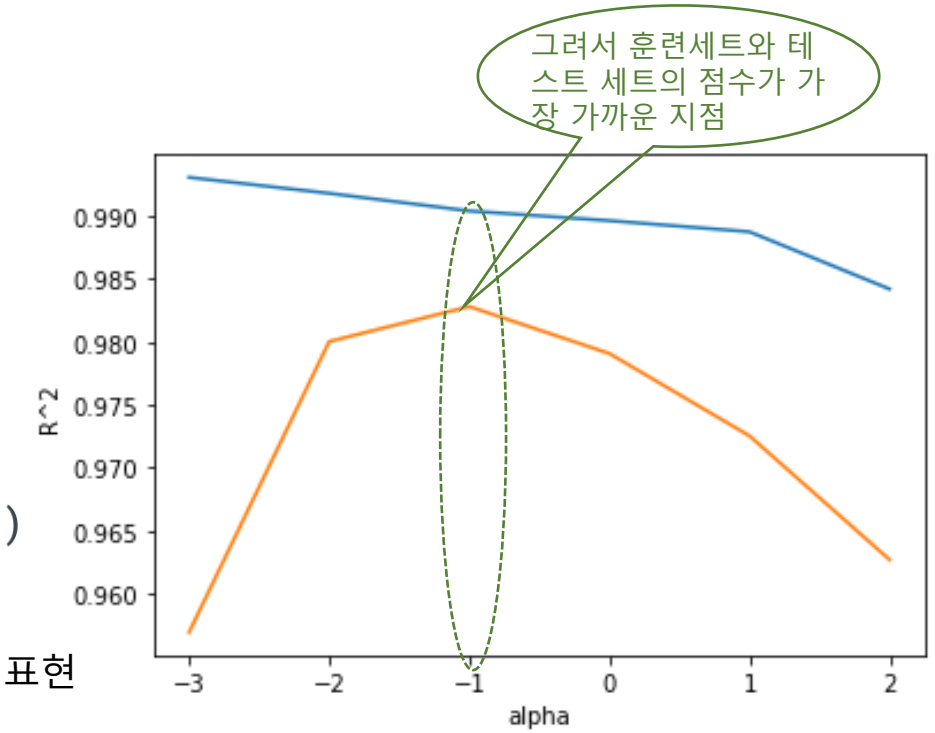
특성에 곱해지는 계수(기울기) 크기를 작게 만드는 일

```
from sklearn.linear_model import Ridge

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

print(ridge.score(train_scaled, train_target))
0.989610167103734 3

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

• 선형회귀모델 규제

➢ 적절한 규제 강도 찾기

```
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)->로그함수로 바꾸어 지수 표현
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.show()
```

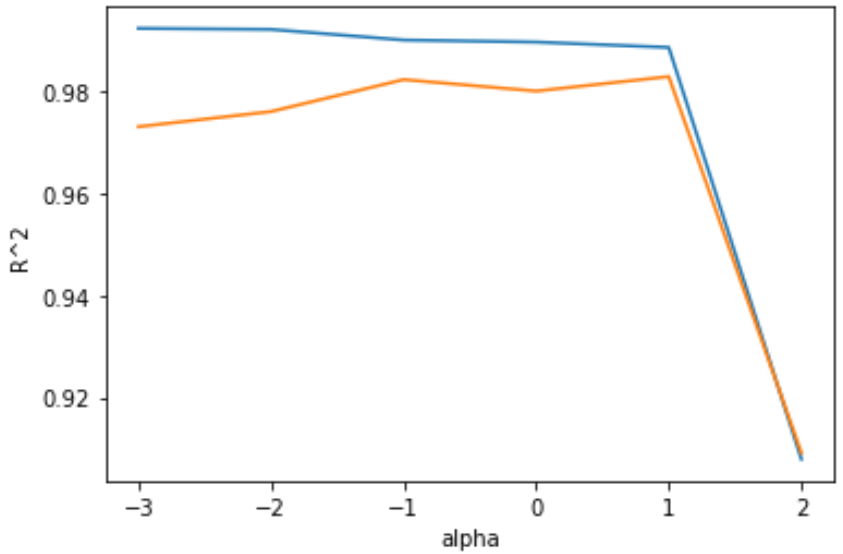



```
ridge = Ridge(alpha=0.1)
ridge.fit(train_scaled, train_target)

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

• 선형회귀모델 규제

➢ 라쏘 회귀 모델

```
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
```




```
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
```

← 계수값이 0인 특성 개수