

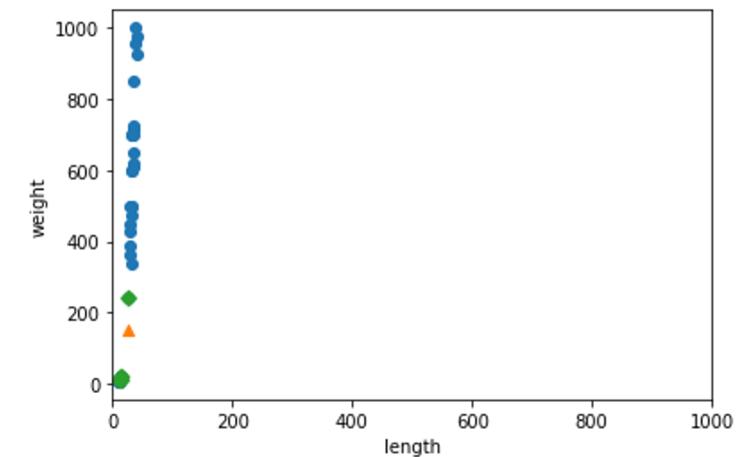
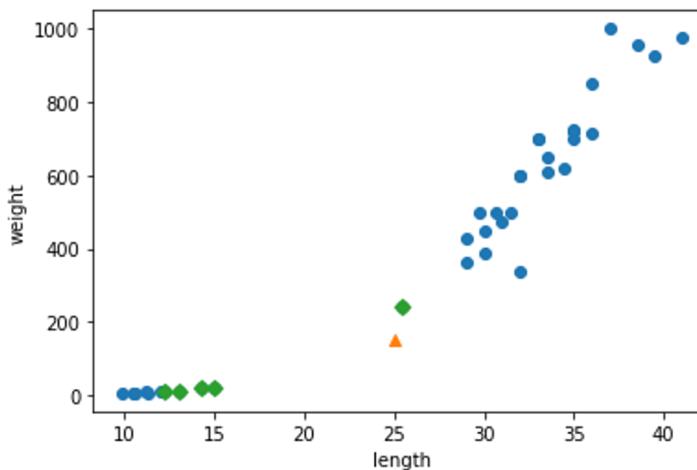
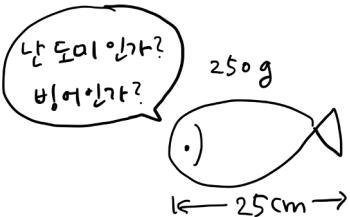
회귀 알고리즘 (1): K-최근접 이웃 회귀, 선형회귀, 릿지, 라쏘회귀

임 경 태

CONTENTS

- 1 복습
- 2 회귀와 K-최근접 이웃회귀
- 3 선형회귀
- 4 다항회귀
- 5 다중회귀

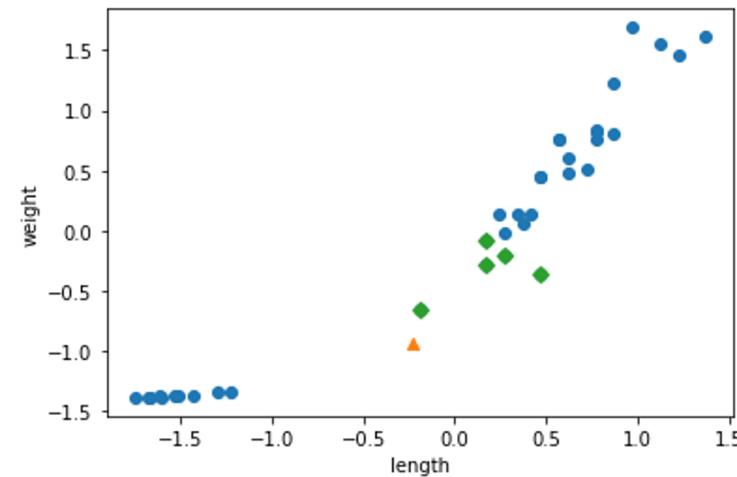
지난 시간 되돌아보기



```
mean = np.mean(train_input, axis=0)
std = np.std(train_input, axis=0)

print(mean, std)
[ 27.29722222 454.09722222] [  9.98244253 323.29893931]

train_scaled = (train_input - mean) / std
```



CONTENTS

1

복습

2

회귀와 K-최근접 이웃회귀

3

선형회귀

4

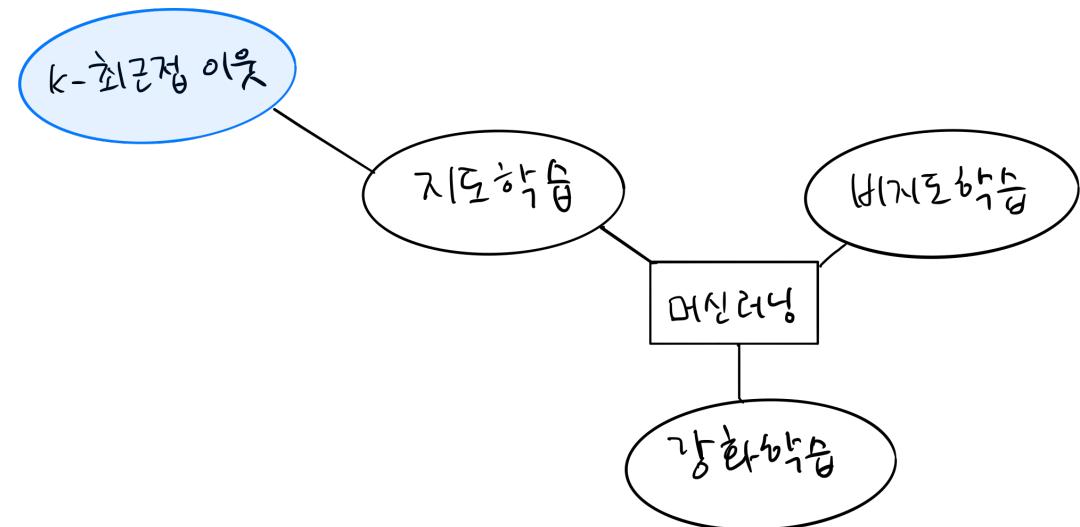
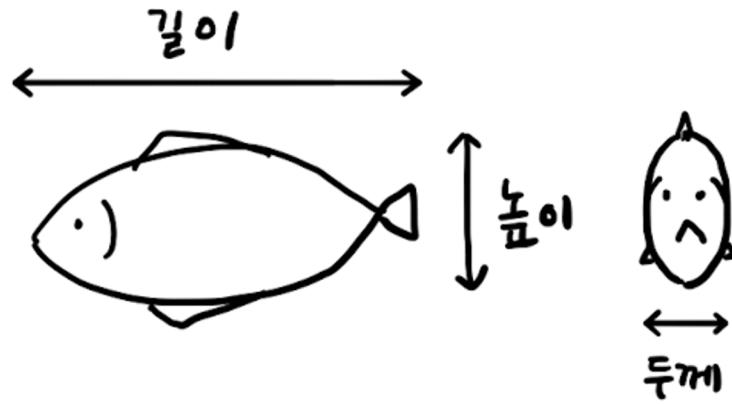
다항회귀

5

다중회귀

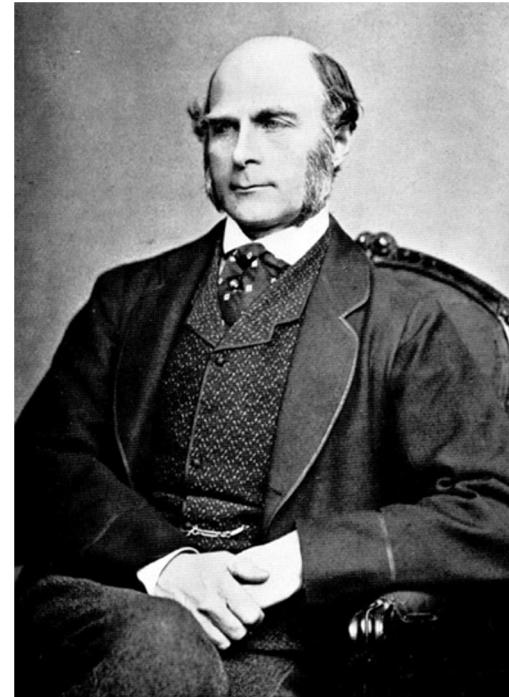
❶ 생선의 무게도 예측할 수 있지 않을까?

- 입력: 길이, 높이, 두께
- 출력: 무게
- 모델: ?



❶ 두 변수 사이의 상관관계를 분석하는 방법

- 회귀의 기원: 키가 큰 사람의 아이는 부모보다 더 크지않다 따라서, **평균으로 회귀**한다

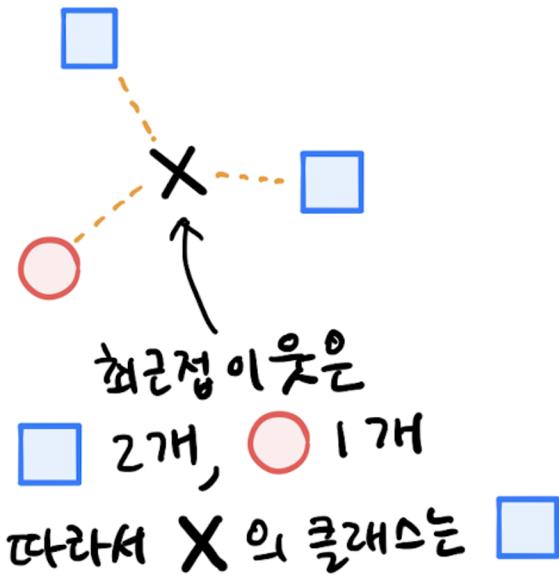


프랜시스 골턴 형님

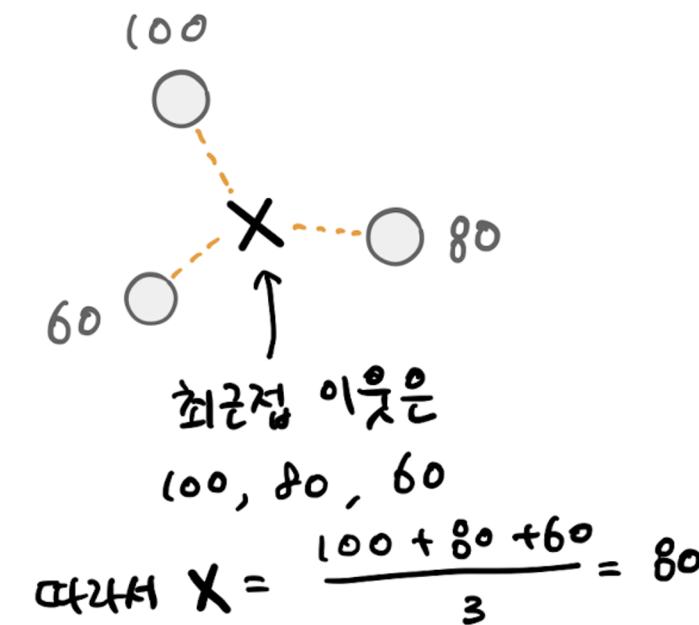
K-최근접 회귀는 무엇인가?

✎ 예측하려는 샘플과 가장 가까운 샘플 K개의 평균 회귀

k-최근접 이웃 분류



k-최근접 이웃 회귀



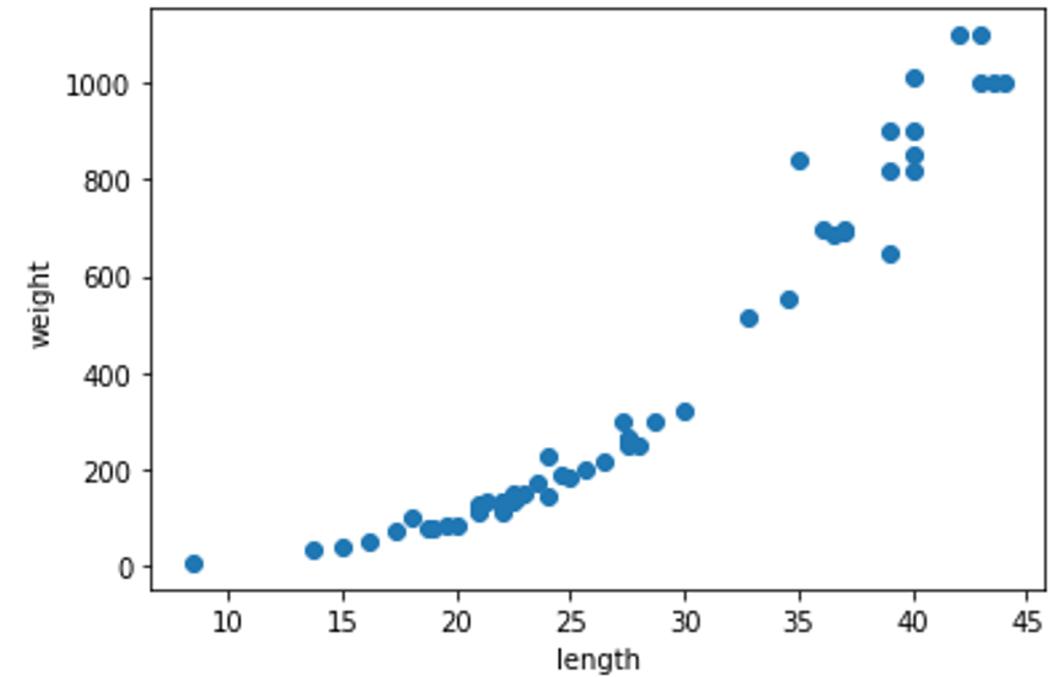
회귀를 어떻게 적용하는가? 1. 가설 설정

회귀: 두 변수 사이의 상관관계를 분석하는 방법

- 가설: 농어의 길이는 무게와 어떤 상관관계가 있을까? (길이가 길면 무거울까?)
- 두 변수: 길이-무게

```
import matplotlib.pyplot as plt

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



회귀를 어떻게 적용하는가? 2. 데이터 준비

- Sklearn에서 훈련데이터는 2차원 배열이어야함.

```
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)  
  
train_input = train_input.reshape(-1, 1)  
test_input = test_input.reshape(-1, 1)
```

[[1],
 [2],
 [3]]
→
[1, 2, 3] →
크기: (3,)
크기: (3, 1)

회귀를 어떻게 적용하는가? 3. 모델 학습 및 평가

fit함수를 이용한 학습 및 결정 계수 기반 평가

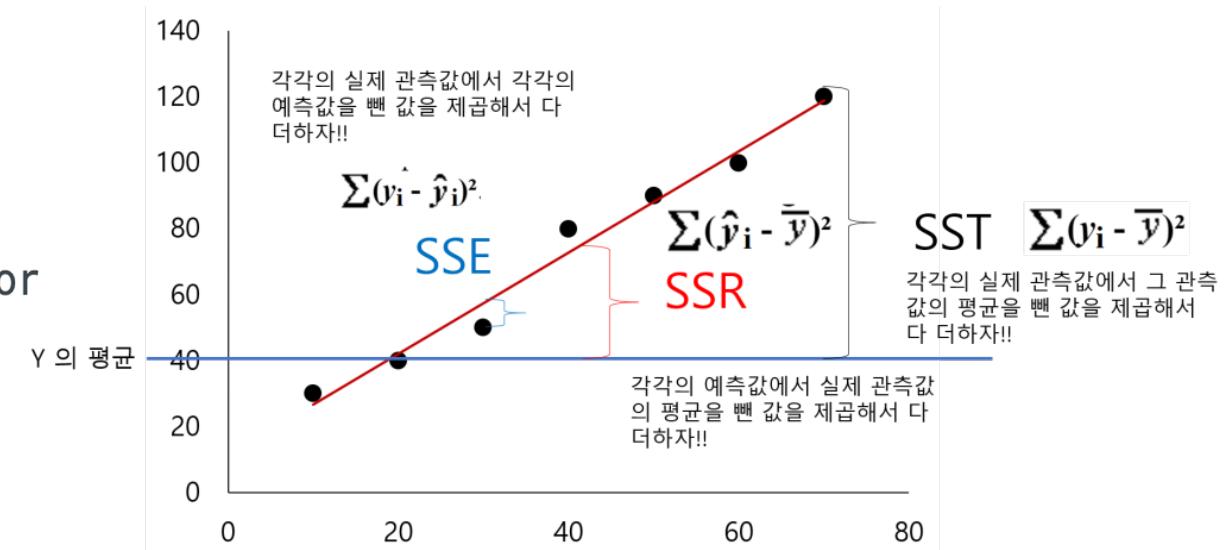
- 결정계수: 회귀모형 내에서 설명변수 x (input)로 설명할 수 있는 반응변수 y (target)의 변동 비율로 성능 측정에 사용됨. 예측이 타깃값과 가까워 질수록 (정답에 가까울 수록) 1에 가까워지고 반대로 예측이 분포의 평균 값 정도를 예측하면 0에 가까워진다. 즉 1에 가까울 수록 좋음

$$R^2 = 1 - \frac{\text{타깃 - 예측}^2 \text{의 합}}{\text{타깃 - 평균}^2 \text{의 합}}$$

```
from sklearn.neighbors import KNeighborsRegressor
```

```
knr = KNeighborsRegressor()  
knr.fit(train_input, train_target)
```

```
knr.score(test_input, test_target)  
0.9928094061010639
```



$$SST = SSR + SSE$$

fit함수를 이용한 학습 및 결정 계수 기반 평가

- Mean Absolut Error (MAE): 전체 샘플에 대해 예측한 값이 실제 정답 값과 얼마나 차이가 있는가?
- $MAE = \sum_{i=1}^n |y - \hat{y}|$

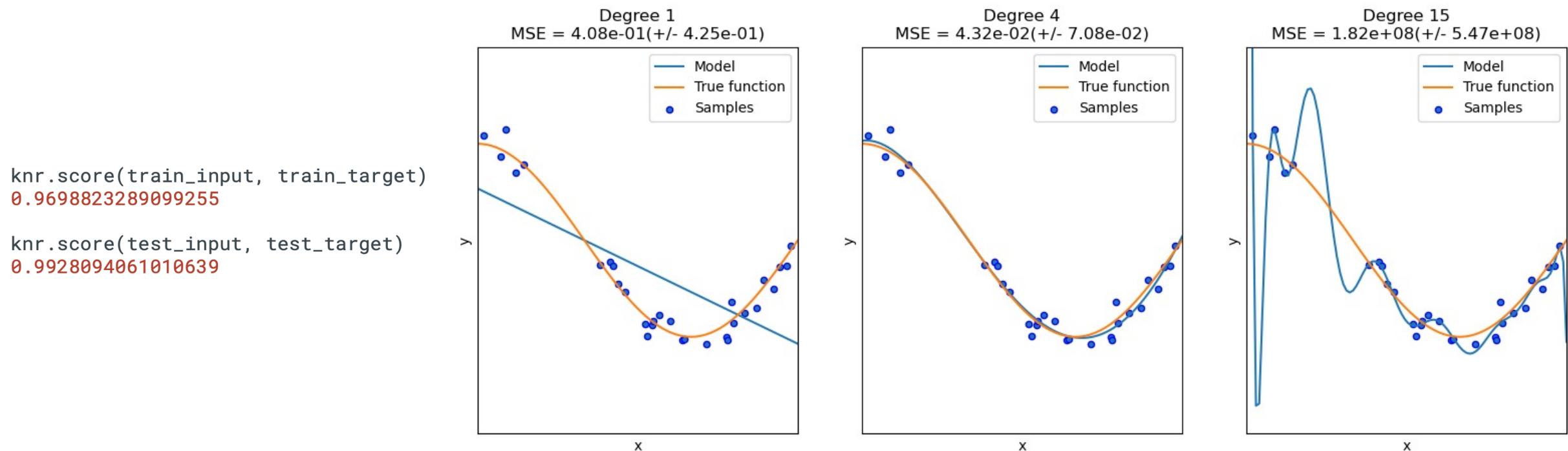
```
from sklearn.metrics import mean_absolute_error  
  
test_prediction = knr.predict(test_input)  
mae = mean_absolute_error(test_target, test_prediction)  
print(mae)
```

19.157142857142862

예측과 실제 정답이 평균
19.15만큼 차이남

📝 과대 적합 (overfitting) 과 과소 적합 (underfitting)

- 과대적합: 학습데이터에 너무 의존함 (학습데이터의 성능이 높고 평가데이터의 성능이 낮음)
- 과소적합: 모델이 너무 단순해 데이터 표현 못함 (평가데이터가 학습데이터보다 성능 높거나 둘다 낮음)

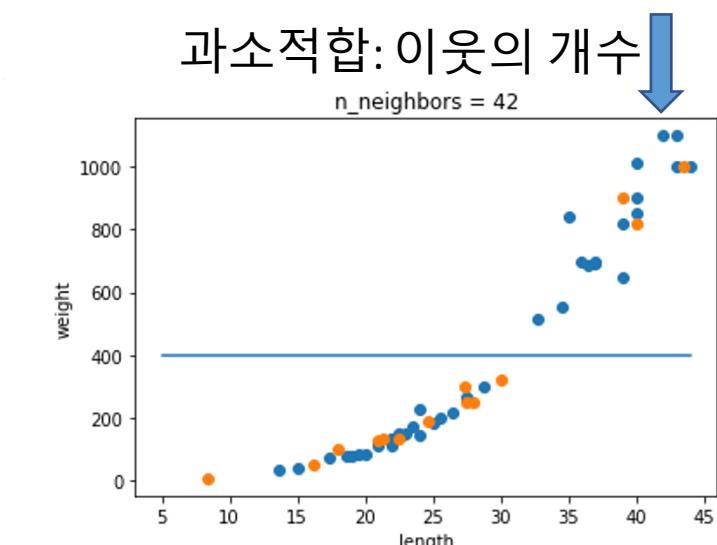
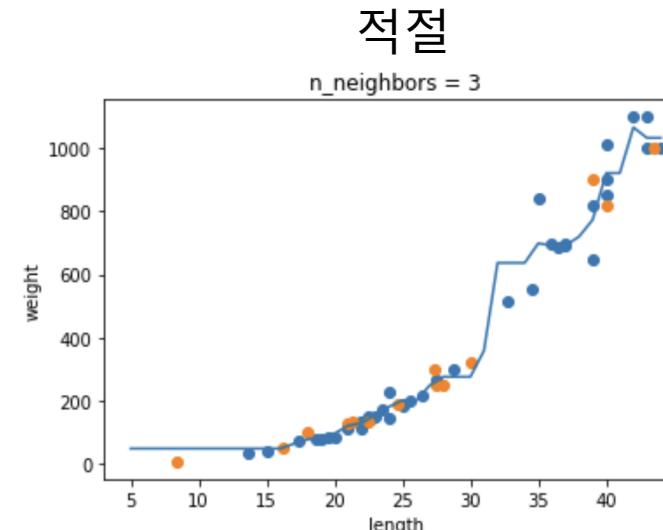
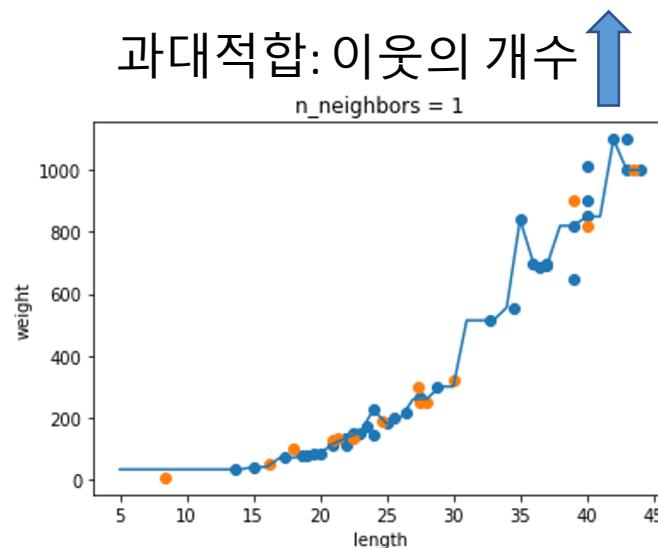


회귀를 어떻게 적용하는가? 5. 분석 결과 피드백

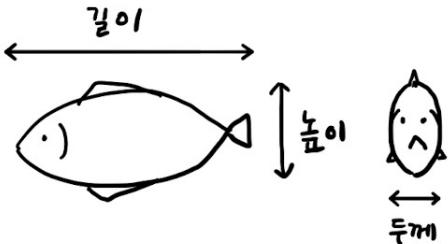
📝 과대 적합 (overfitting)과 과소 적합 (underfitting) 해결

- K-NN에서 K값을 감소시키면: 주변 가까운 것만 참조, 즉 지역성이 올라감 (민감해 짐=모델이 복잡해 짐)
- K-NN에서 K값을 증가시키면: 전체 데이터를 고려, 일반성이 올라감 (두리뭉실 평균값을 모사 하려함)

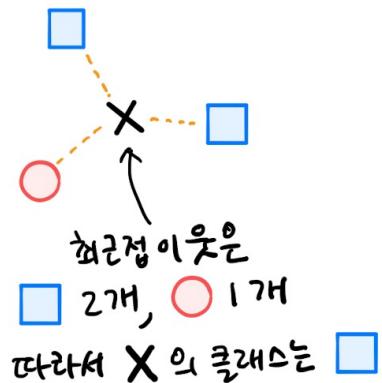
```
knr.n_neighbors = 3  
knr.fit(train_input, train_target)  
  
print(knr.score(train_input, train_target))  
0.9804899950518966  
  
print(knr.score(test_input, test_target))  
0.974645996398761
```



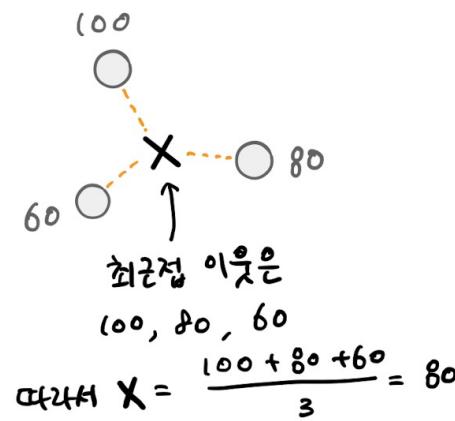
K-최근접 회귀 정리



k-최근접 이웃 분류



k-최근접 이웃 회귀



$$R^2 = 1 - \frac{\text{(타겟 - 예측)}^2 \text{의 합}}{\text{(타겟 - 평균)}^2 \text{의 합}}$$

`knr.score(train_input, train_target)`

0.9698823289099255

`knr.score(test_input, test_target)`

0.9928094061010639

과대적합



이웃의 개수



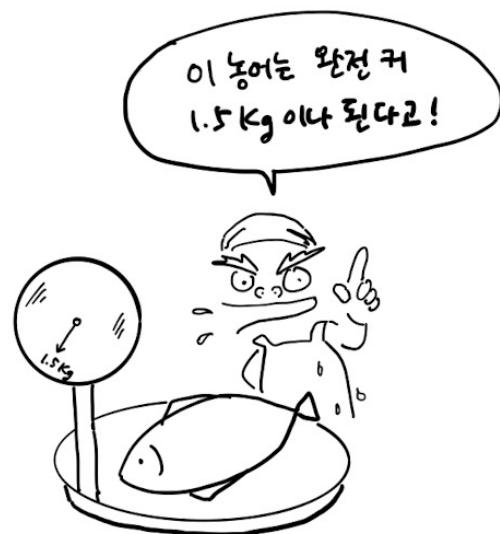
과소적합

CONTENTS

- 1 복습
- 2 회귀와 K-최근접 이웃회귀
- 3 선형회귀
- 4 다항회귀
- 5 다중회귀

❶ K-근접 이웃 회귀의 문제점

- 사례기반 학습 방법으로 학습데이터의 값의 분포를 활용하여 예측하기 때문에 outlier에 대한 예측에 취약함

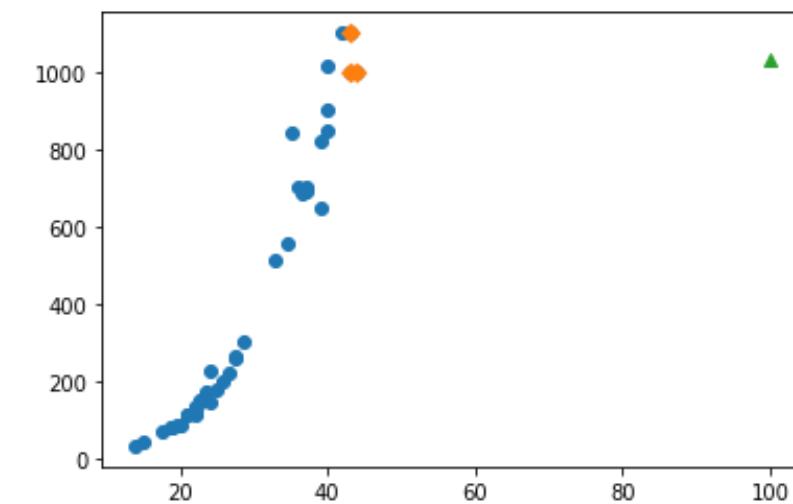
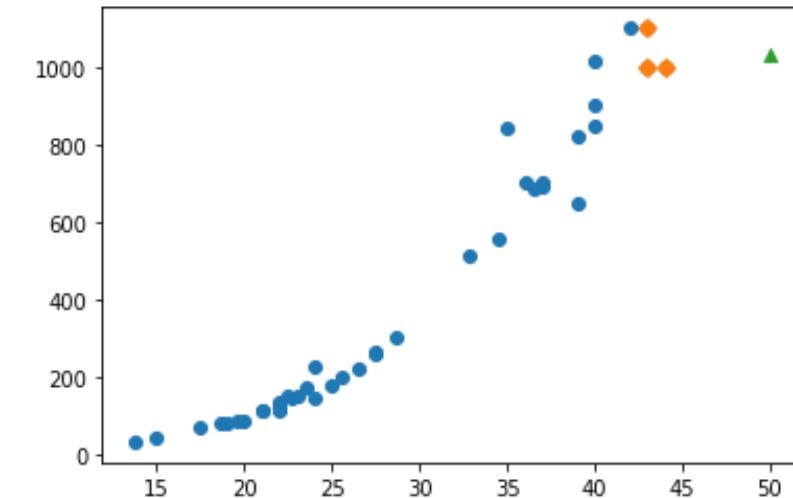


```
print(knr.predict([[50]]))  
[1033.33333333]
```

📝 K-근접 이웃 회귀의 문제점

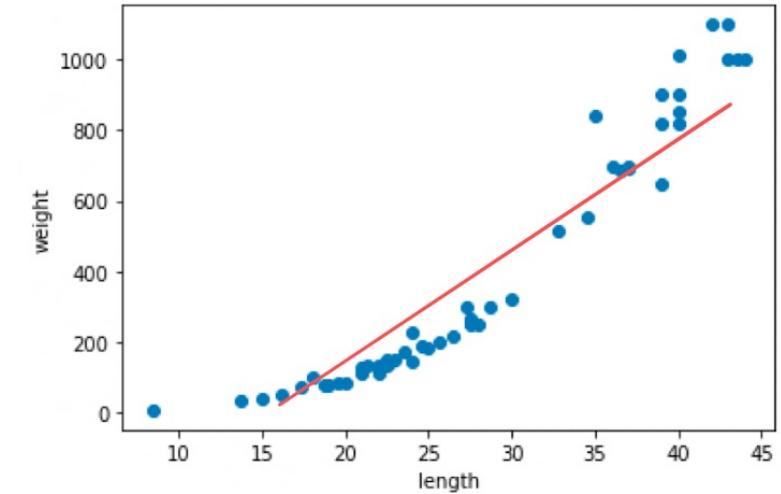
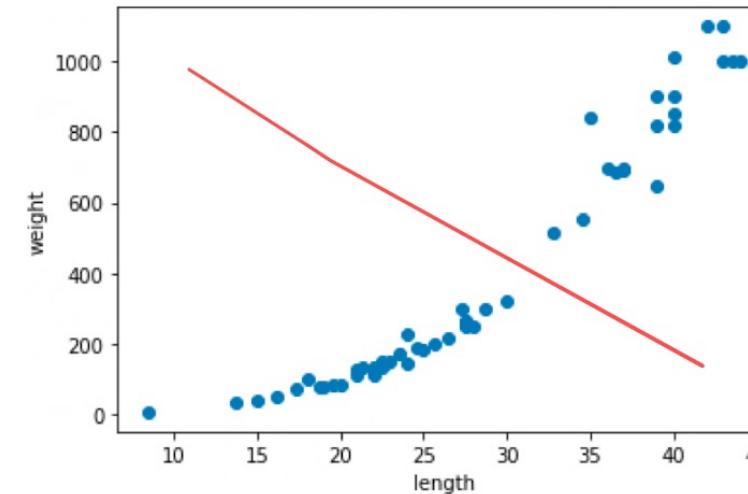
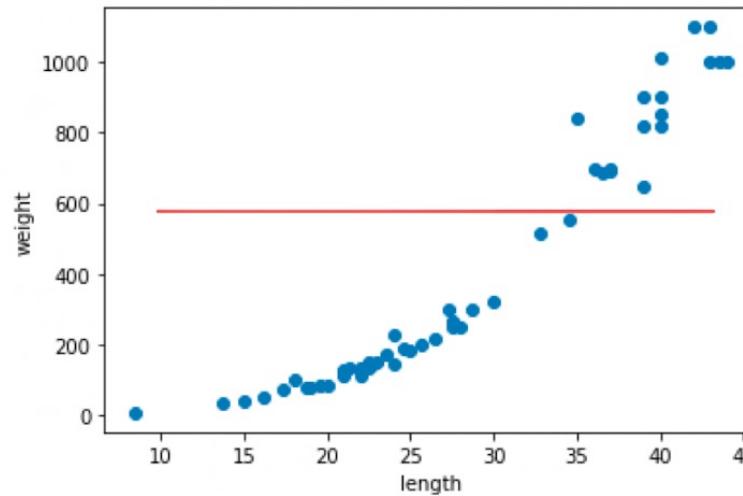
- 50 cm 농어의 이웃과 100 cm 농어의 이웃이 같음. 따라서 같은 무게가 예측됨

```
# 50cm 농어의 이웃을 구합니다  
distances, indexes = knr.kneighbors([[50]])  
  
# 훈련 세트의 산점도를 그립니다  
plt.scatter(train_input, train_target)  
# 훈련 세트 중에서 이웃 샘플만 다시 그립니다  
plt.scatter(train_input[indexes], train_target[indexes],  
            marker='D')  
# 50cm 농어 데이터  
plt.scatter(50, 1033, marker='^')  
plt.show()
```



데이터의 관계를 선형으로 표현할 수 있지 않을까?

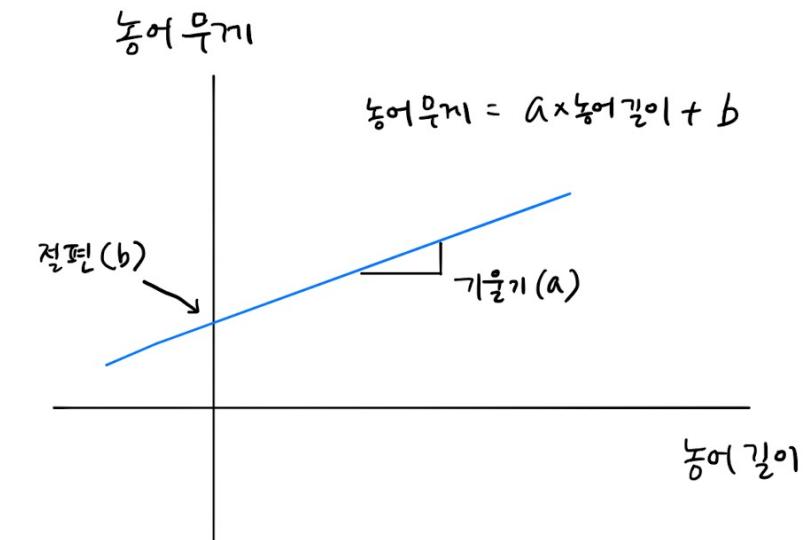
- 종속 변수 y 와 한 개 이상의 독립 변수 X 와의 선형 상관 관계를 모델링하는 회귀분석 기법 (위키)
- 예를 들어, $y = 3x + 5$ 라는 함수가 있을 때 y 는 x 에 따라 값이 변하므로 종속변수이고, x 는 독립변수이다.



선형회귀의 목적

데이터를 표현할 수 있는 최적의 기울기와 절편 값을 찾는다.

```
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
```



선형회귀의 목적

❶ 데이터를 표현할 수 있는 최적의 기울기와 절편 값을 찾는다.

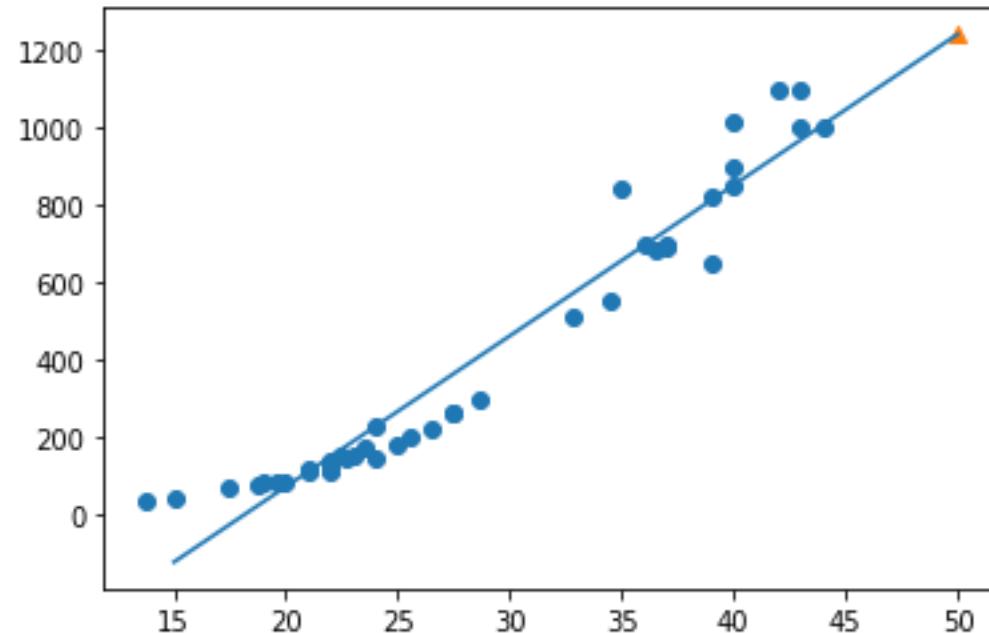
```
# 훈련 세트의 산점도를 그립니다  
plt.scatter(train_input, train_target)
```

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

```
# 50cm 높어 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
```

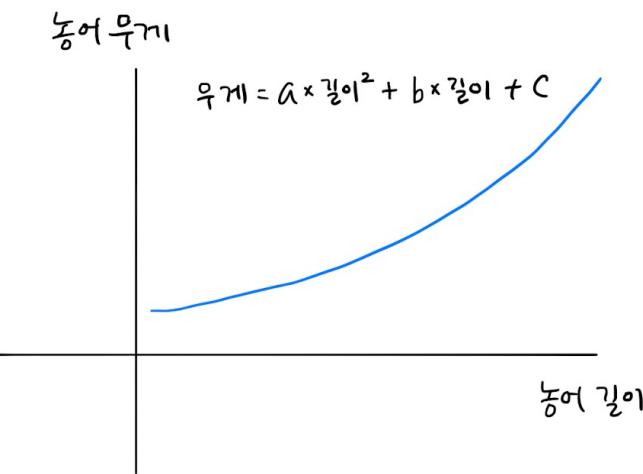


CONTENTS

- 1 복습
- 2 회귀와 K-최근접 이웃회귀
- 3 선형회귀
- 4 다항회귀
- 5 다중회귀

📝 항이 여러 개인 함수를 이용한 회귀분석

- 항은 제곱근이나 2차, 3차항 등 다양하게 있으며 함수의 형태가 비선형이라는 특징이 있음.
- It is a special case of linear regression, by the fact that we create some polynomial features before creating a linear regression.



7세기

384.16	19.6
484	22
349.69	18.7
:	:
1190.25	34.5

42

2

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

다항회귀의 학습

```
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
```

$$\hat{y} = 1.01 \times 50^2 - 21.55792498 + 116.05$$

📝 항이 여러 개인 함수를 이용한 회귀분석

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

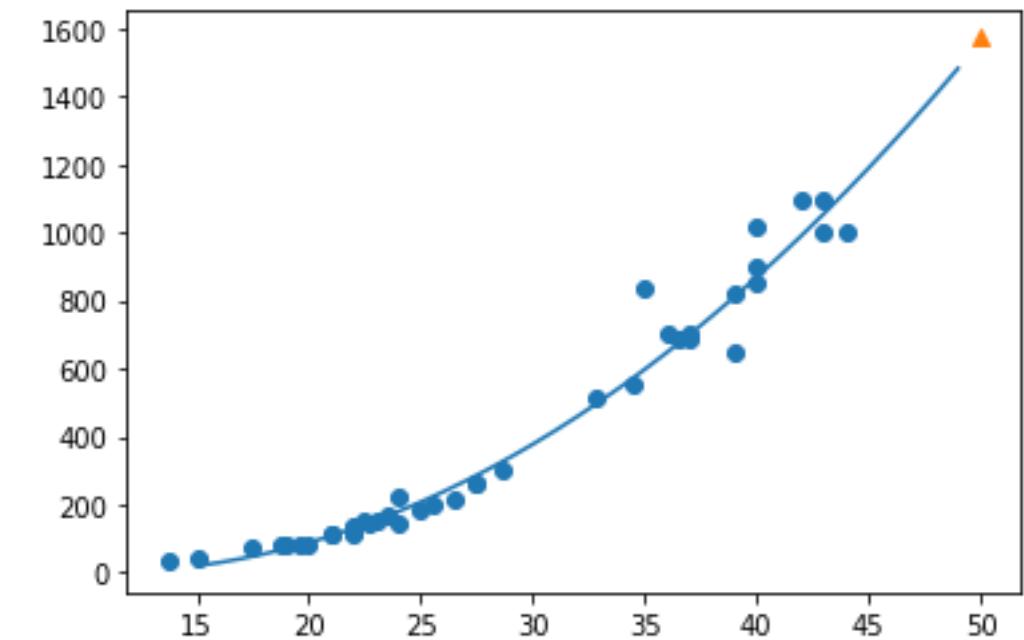
```
# 훈련 세트의 산점도를 그립니다  
plt.scatter(train_input, train_target)
```

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

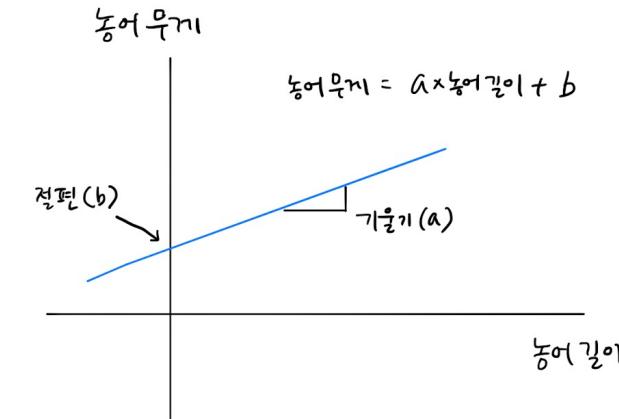
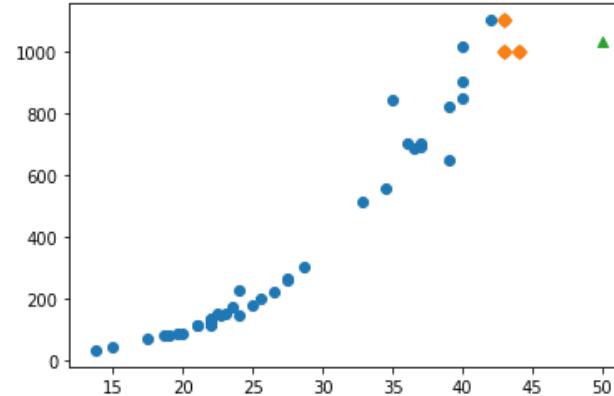
```
# 50cm 놓어 데이터  
plt.scatter([50], [1574], marker='^')  
plt.show()
```

```
print(lr.score(train_poly, train_target))  
0.9706807451768623
```

```
print(lr.score(test_poly, test_target))  
0.9775935108325122
```



선형회귀 정리

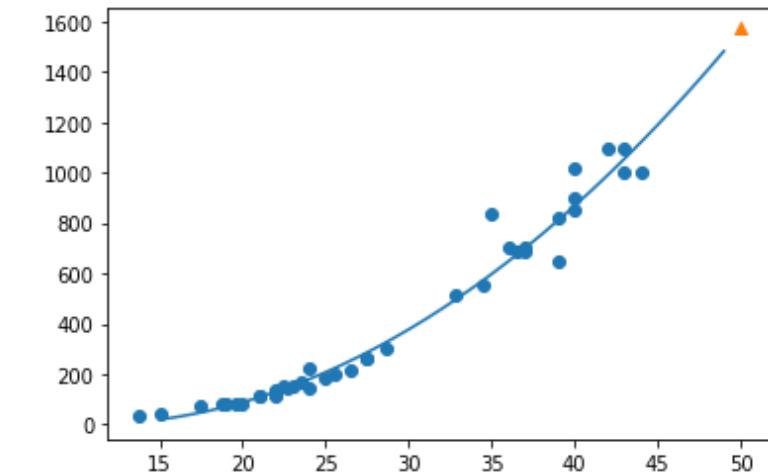


```
from sklearn.linear_model import LinearRegression  
  
lr = LinearRegression()  
# 선형 회귀 모델 훈련  
lr.fit(train_input, train_target)  
  
# 50cm 농어에 대한 예측  
print(lr.predict([[50]]))  
[1241.83860323]  
  
print(lr.coef_, lr.intercept_)  
[39.01714496] -709.0186449535477
```

7세공

384.16	19.6
484	22
349.69	18.7
:	:
1190.25	34.5

2



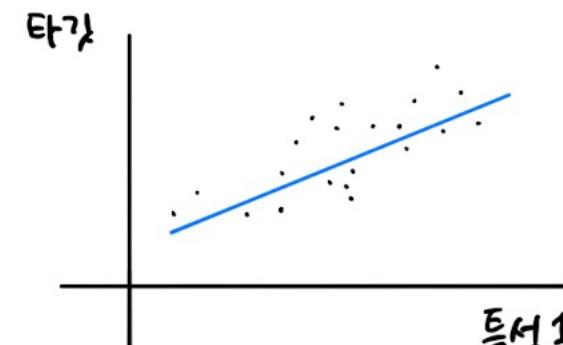
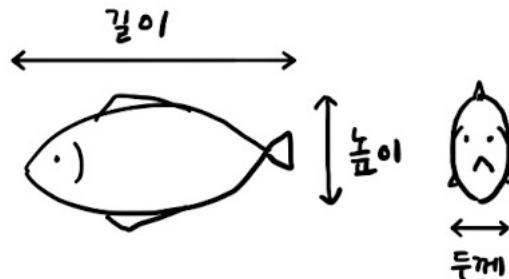
CONTENTS

- 1 복습
- 2 회귀와 K-최근접 이웃회귀
- 3 선형회귀
- 4 다항회귀
- 5 다중회귀

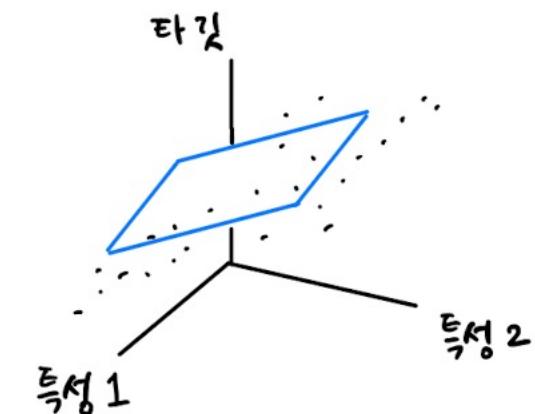
다중회귀 (multiple regression)

❶ 독립변수가 2개 이상인 경우를 분석 대상으로 하는 회귀분석

- 농어의 길이 뿐만 아니라 높이 두께도 함께 사용할 경우의 선형회귀
- 특성이 2개일 경우 회귀 직선은 직선이 아닌 하나의 평면을 학습함



$$y = ax + b$$

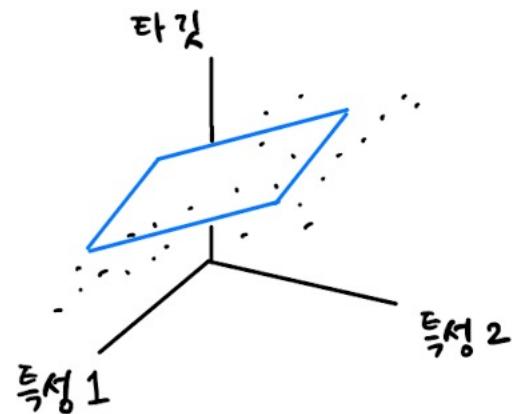


$$z = ax + by + c$$

특성공학 (Feature Engineering)

❶ 독립변수가 2개 이상일 때도 새로운 특성을 추가할 수 있지 않을까?

- **특성공학**: 기존의 특성을 이용해 새로운 특성을 뽑아내는 작업
- 예) 농어높이 x 농어 길이, 농어무게 x 농어 두께… 등



$$z = ax + by + c$$

$$z = ax + by + c + dx^2 + ey^2 + fxy$$

$$z = ax + by + c + dx^2 + ey^2 + fxy + gx^3 + hy^3 + ix^2y^2$$

학습과정 (1): 데이터 읽기

CSV 파일

length	height	width
8.4	2.11	1.41
13.7	3.53	2.0
:		

→ 판다스 데이터프레임 네이미 배열
pd.read_csv() → to_numpy()

```
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 ]]
```

학습과정 (2): PolynomialFeatures를 이용한 특성공학 적용

```
from sklearn.preprocessing import PolynomialFeatures  
  
# degree=2  
poly = PolynomialFeatures()  
poly.fit([[2, 3]])  
  
# 1(bias), 2, 3, 2**2, 2*3, 3**2  
print(poly.transform([[2, 3]]))  
[[1. 2. 3. 4. 6. 9.]]
```

학습과정 (3): PolynomialFeatures를 이용한 특성공학 적용

```
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)
```

```
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
```

학습과정 (4): PolynomialFeatures를 이용한 더 많은 특성공학 적용

특성공학을 이용해 특성을 5제곱 까지 늘려보자

```
poly = PolynomialFeatures(degree=5, include_bias=False)

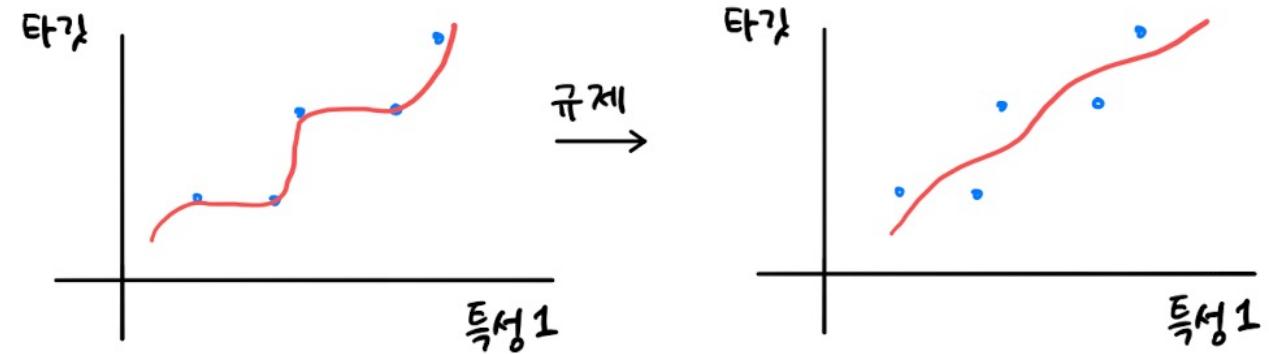
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)

print(train_poly.shape)
(42, 55)

lr.fit(train_poly, train_target)

print(lr.score(train_poly, train_target))
0.999999999991097

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

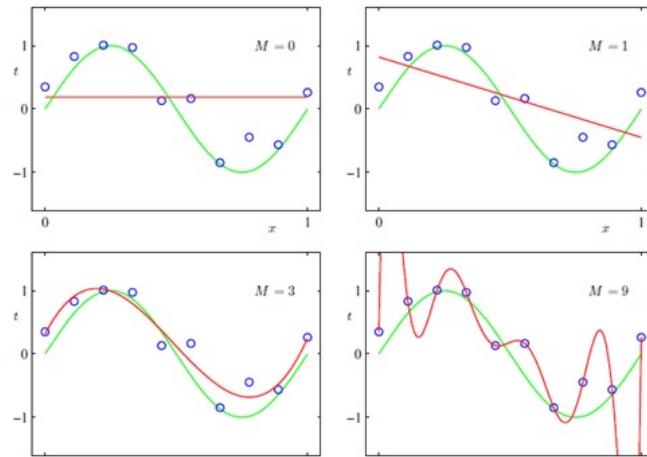


과대적합 발생: 특성의 개수를 과하게 늘리면 학습 데이터의 미세한 변화에도 민감하게 반응하는 모델이 학습됨.
이를 위해 특성에 곱해지는 계수(기울기)의 크기를 작게 만들어야 함 (혹은 학습데이터를 증가 시켜야 함).

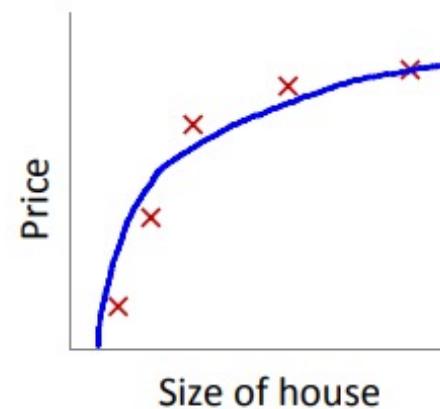
규제 (Regularization)

☞ 규제: 모델이 훈련데이터에 overfit하게 되는 것을 방해

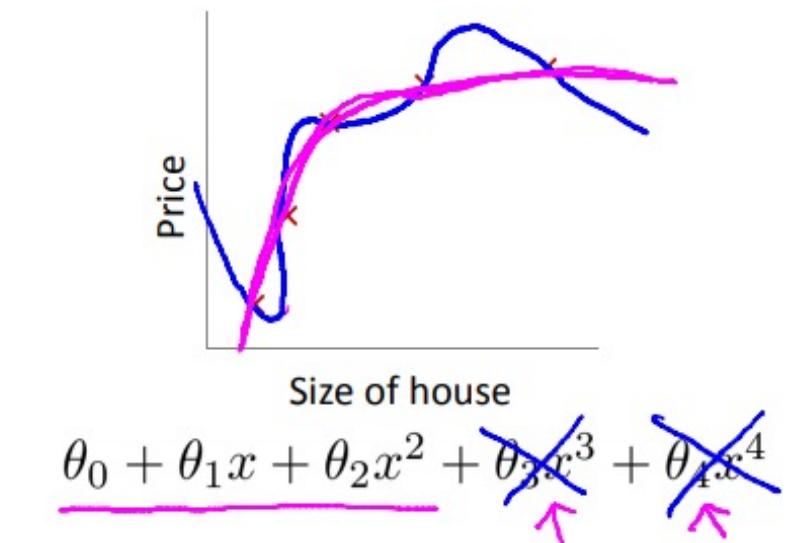
- 높은 차원의 계수(기울기)값을 0에 가깝게 만든다면 더 낮은 차원의 함수로 변환, smooth한 예측 가능



$$y = 0.35 + 232.37 * x - 5321.83 * x^2 + 48568.31 * x^3 - 231639.30 * x^4 + 640042.26 * x^5 - 1061800.52 * x^6 + 1042400.18 * x^7 - 557682.99 * x^8 + 1252201.43 * x^9 \quad \text{-(Eq 1.3)}$$



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\underline{\theta_0 + \theta_1 x + \theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

✎ 기울기(모델 파라미터)의 값이 커지는것을 방지한 회귀모델

$$\min_w \sum_{i=1}^n \left(y_i - \hat{y}_i \right)^2 + 10000w_0^2 + 10000w_1^2$$

```
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
```

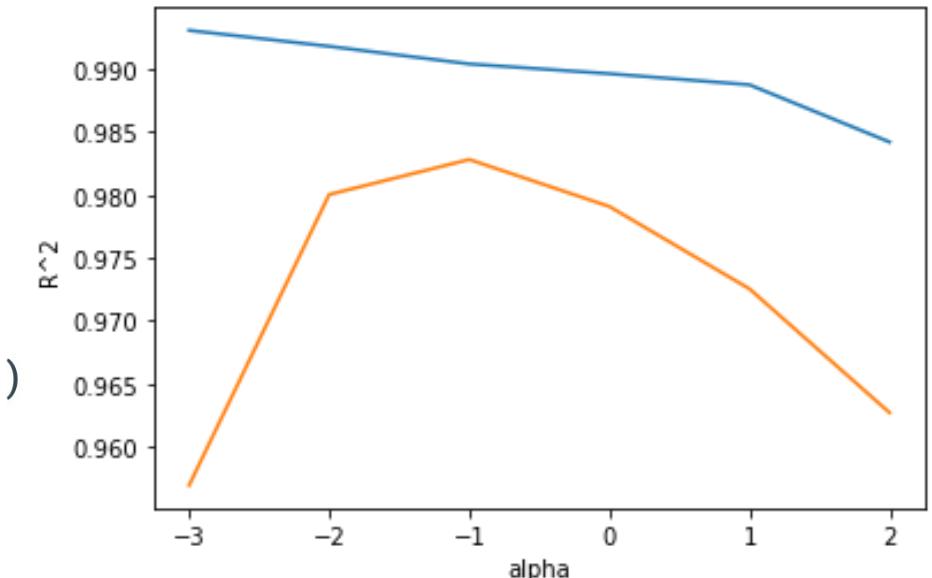
릿지 회귀의 표준화

```
from sklearn.preprocessing import StandardScaler  
  
ss = StandardScaler()  
ss.fit(train_poly)  
  
train_scaled = ss.transform(train_poly)  
test_scaled = ss.transform(test_poly)
```

☞ 릿지회귀에서 규제는 얼마나 적용해야 할까?

```
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()
```



```
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
```

✎ 기울기(모델 파라미터)의 값이 커지는것을 방지한 회귀모델

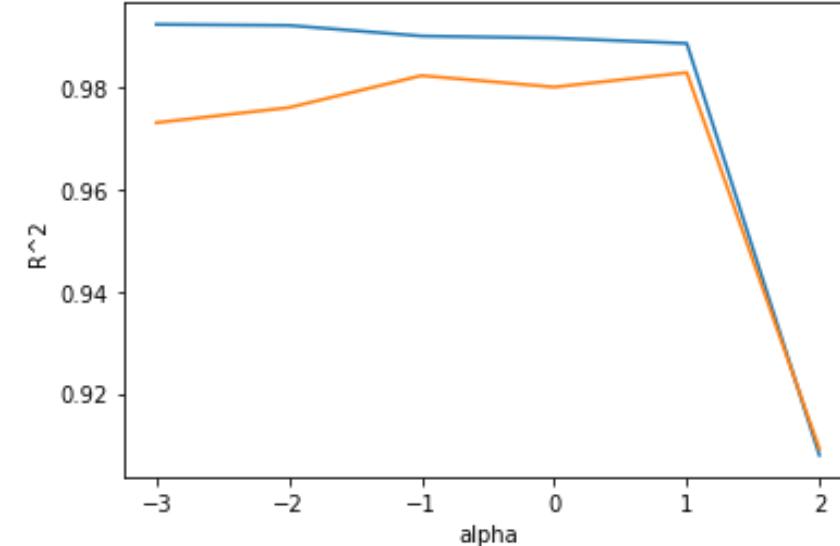
$$\min_w \sum_{i=1}^n \left(y_i - \hat{y}_i \right)^2 + 10000|w_0| + 10000|w_1|$$

```
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
```

감사합니다.