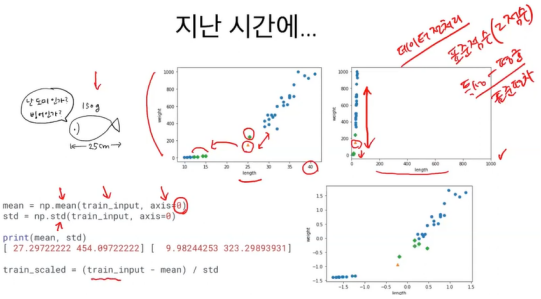
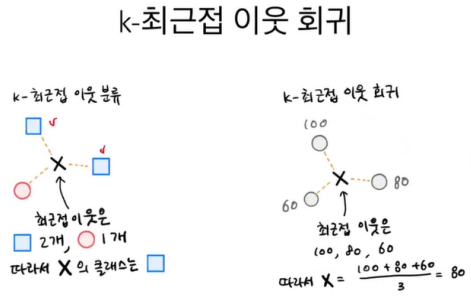
week2. 회귀 알고리즘과 모델

▼ week 1 복습

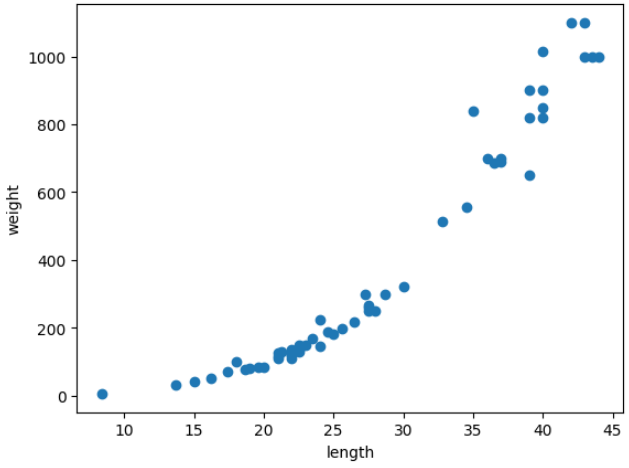


K- 최근접 이웃 회귀

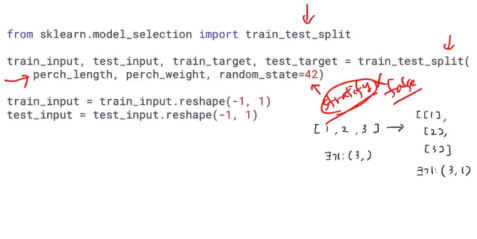



```
import numpy as np
perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 1
       21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5,
       23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3,
       27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0,
       39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0,
       44.0])
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0,
```

```
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1
       1000.0])

import matplotlib.pyplot as plt
plt.scatter(perch_length, perch_weight : 타깃. )
plt.xlabel('length')
plt.ylabel('weight')
plt.show
```



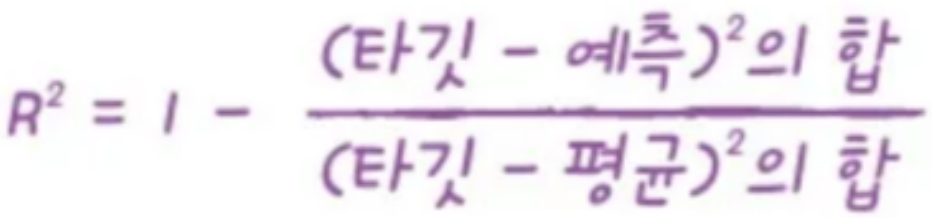
- 배열 크기 바꾸기
- 회귀에서는 stratify 를 잘 사용하지 않는다.




```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_te
```

```
    perch_length, perch_weight, random_state=42
)

test_array = np.array([1,2,3,4])
print(test_array.shape) # (4,) 배열을 (2,2) 크기로 바꾸기

test_array = test_array.reshape(2,2)
print(test_array.shape)

# 지정한 크기와 원본 배열의 원소 개수가 다르면 에러 발생.

# train_input 의 크기는 (42, ), 2차원 배열인 (42,1) 로 바꾸기 위해 t
# 크기에 -1을 지정하면, 나머지 원소 개수로 모두 채우라는 의미.
# 예시로, 첫 번째를 나머지 원소 개수로 채우고, 두 번째 크기를 1로 하려면 t
train_input= train_input.reshape(-1,1)
test_input = test_input.reshape(-1,1)
print(train_input.shape, test_input.shape)
```

결정계수 (R^2)

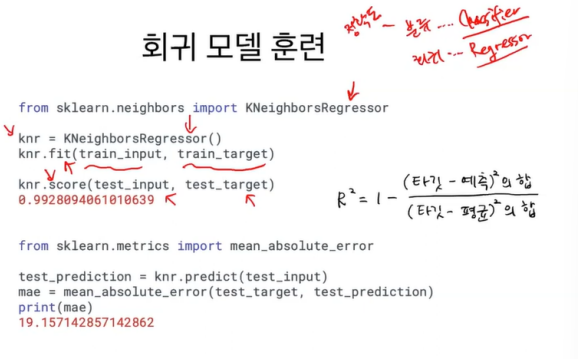



```
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor()

# k-최근접 이웃 회귀 모델 훈련 : 회귀에서는 정확한 숫자를 맞히는 것이 거의
```

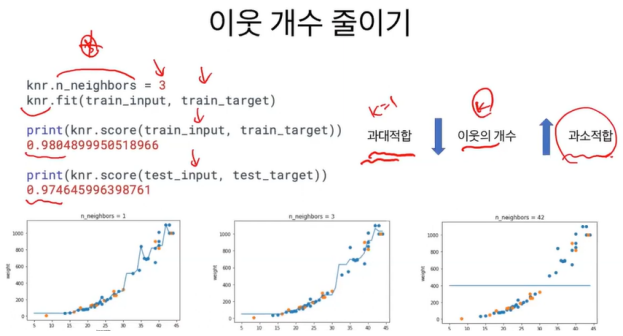
```
# 이를 '결정계수' 라고 부름, 간단히 R^2 라고 함.
knr.fit(train_input, train_target)
print(knr.score(test_input, test_target))
```

💡 mean_absolute_error : 타킷과 예측의 절댓값 오차를 평균하여 반환.

```
# score() 메서드의 출력값의 의미 : 출력값이 높을수록 좋은 것. 정확도, 결
# 만약, score() 메서드가 에러율을 반환한다면 실제로는 낮은 에러가 높은 값

# 사이킷런 패키지 : sklearn.metrics 패키지 아래 여러 측정 도구
# mean_absolute_error : 타깃과 예측의 절댓값 오차를 평균하여 반환
from sklearn.metrics import mean_absolute_error

# 테스트 세트에 대한 예측을 만든다.
test_prediction= knr.predict(test_input)

# 테스트 세트에 대한 평균 절댓값 오차를 계산한다
mae = mean_absolute_error(test_target, test_prediction)
print(mae)

# 19 = 예측이 평균적으로 19g 정도 타깃값과 다르다는 것을 의미.
```



과대적합 vs 과소적합




```
print(knr.score(train_input, train_target))
# 훈련 세트에서 더 좋은 점수가 나와야 하는데 왜 점수가 더 낮을까 ..?
# 과소 적합 : 너무 단순한 모델 때문.
# 과대 적합 : 훈련 세트에만 잘 맞는 모델.
0.9698823289099254

# n_neighbors 속성값을 바꾸기

# 이웃의 개수를 3으로 설정.
knr.n_neighbors=3

# 모델을 다시 훈련
knr.fit(train_input, train_target)
print(knr.score(train_input, train_target))
0.9804899950518966

print(knr.score(test_input, test_target))
0.9746459963987609
```

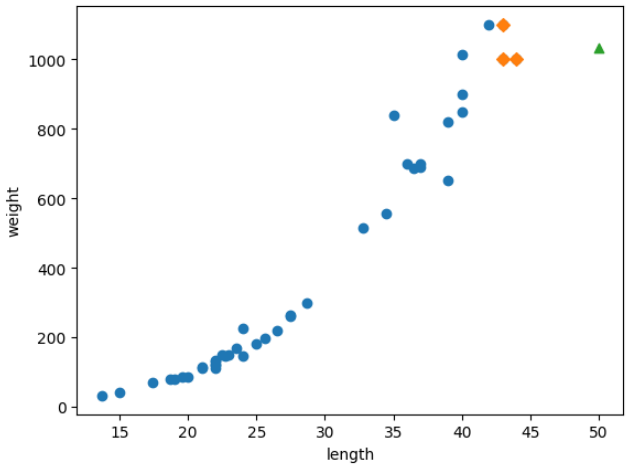
k-최근접 모델의 한계 :

주위에 이웃들을 바탕으로 판단하기 때문에,

엄청나게 큰 값이 입력되었을 때 잘못된 예측을 할 가능성이 높음.

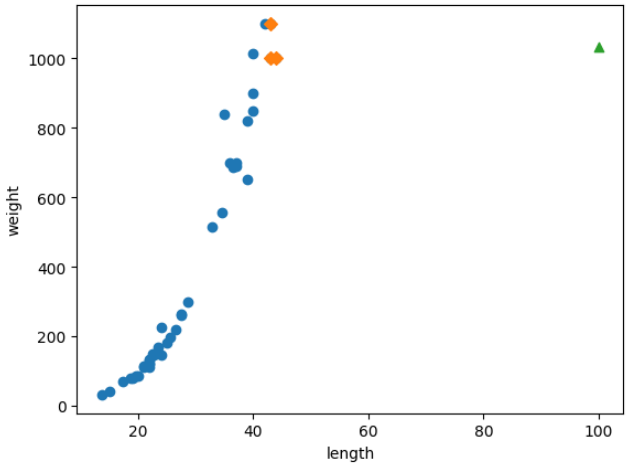
```
import matplotlib.pyplot as plt

# 50cm 농어의 이웃 구하기
distances, indexes = knr.kneighbors([[50]])
```

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

# 훈련 세트 중에서 이웃 샘플만 다시 그리기
plt.scatter(train_input[indexes], train_target[indexes], mark

# 50cm 농어 데이터
plt.scatter(50, 1033, marker='^')
plt.xlabel('lenght')
plt.ylabel('weight')
plt.show()
```

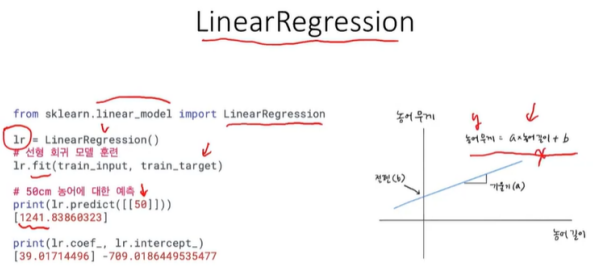


50cm 농어에서 가장 가까운 것은 45cm 농어 근방의 이웃이기에, 엉뚱한 값을 예측한 것이다.

즉, k-최근접 이웃을 이용해 문제를 해결하려면 가장 큰 농어가 포함되도록 훈련 세트를 다시 만들어야 함.

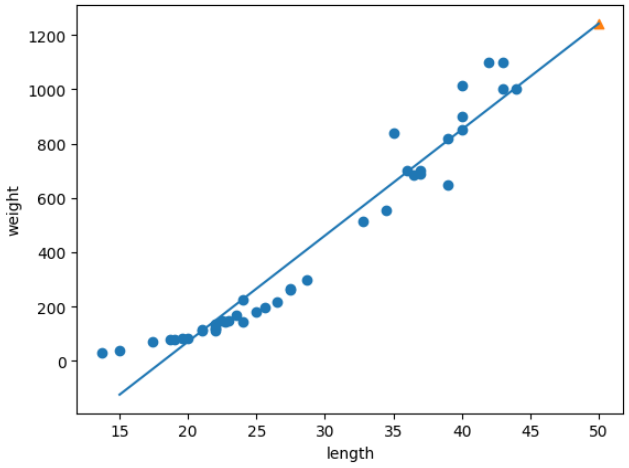


선형 회귀



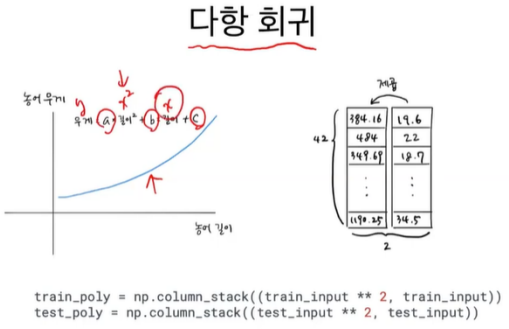

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

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

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

print(lr.coef_, lr.intercept_) # 머신러닝에서는 기울기를 계수 또는 ㄱ

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

# 15에서 50까지 1차 방정식 그래프를 그림
```

```
plt.plot([15, 50], [15*lr.coef_+lr.intercept_, 50*lr.coef_+lr

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



그러나, 선형 모델이 무게를 음수까지 예측할 가능성이 있음. 따라서, 직선보다는 2차 함수 형태가 적합할 것으로 보임.

다항 회귀

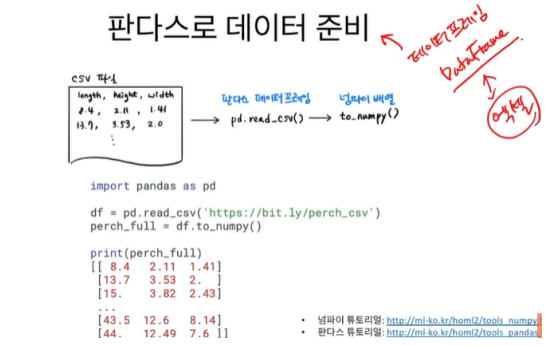


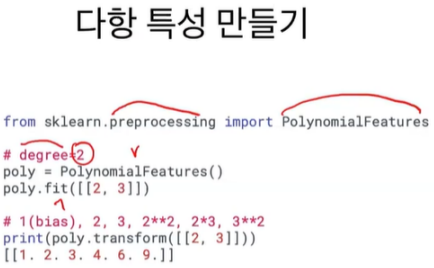

```
train_poly = np.column_stack((train_input **2, train_input))
test_poly = np.column_stack((test_input ** 2, test_input))
```

```
print(train_poly.shape, test_poly.shape)

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

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

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

- `lr.coef_`: 학습된 선형 회귀 모델의 계수. 각 독립 변수(feature)에 해당하는 가중치를 나타내며, 회귀식의 기울기를 의미.
- `lr.intercept_`: 학습된 선형 회귀 모델의 절편입니다. 회귀식에서 y축과 교차하는 값, 즉 모든 독립 변수(feature)가 0일 때 종속 변수의 예측값을 나타냅니다.

다중회귀 , 판다스 : 데이터 준비






```
import pandas as pd # pd는 판다스의 별칭
df = pd.read_csv('https://bit.ly/perch_csv_data')
perch_full=df.to_numpy()
```

```
print(perch_full)

import numpy as np
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1
       1000.0])

from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_te

from sklearn.preprocessing import PolynomialFeatures

# transform 전에 꼭 poly.fit 을 사용해야함. 두개를 하나로 합친 fit_tr
poly = PolynomialFeatures(include_bias=False)
poly.fit([[2, 3]])
print(poly.transform([[2, 3]]))

poly = PolynomialFeatures(include_bias=False)
# PolynomialFeatures 클래스는 특성이 어떻게 만들어졌는지 확인.
# include_bias=False는 상수 항(1)을 포함하지 않도록 설정하는 옵션

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

print(train_poly.shape)

poly.get_feature_names_out()

test_poly = poly.transform(test_input)
```

다중 회귀 모델 훈련

```
from sklearn.linear_model import LinearRegression

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

print(lr.score(test_poly, test_target))

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)

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

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

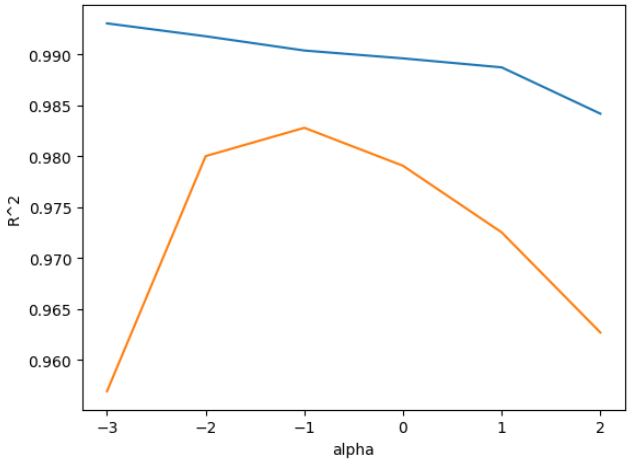
규제 기법 : 릿지 , 리쏘

1. 릿지 회귀 (Ridge Regression)

- 변수를 제거하지 않고 가중치만 줄이기를 원할 때 사용

```
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))

import matplotlib.pyplot as plt
```

```
train_score = []
test_score = []

여기서 알파는 , 하이퍼파라미터라고도 함.
사전에 지정해야 하는 값을 의미함.
적절한 알파 값을 찾기 위하여 R^2 그래프를 그림.
alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
    # 릿지 모델
    ridge = Ridge(alpha=alpha)
    # 릿지 모델 훈련
    ridge.fit(train_scaled, train_target)
    # 훈련 점수와 테스트 점수를 저장
    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()
```

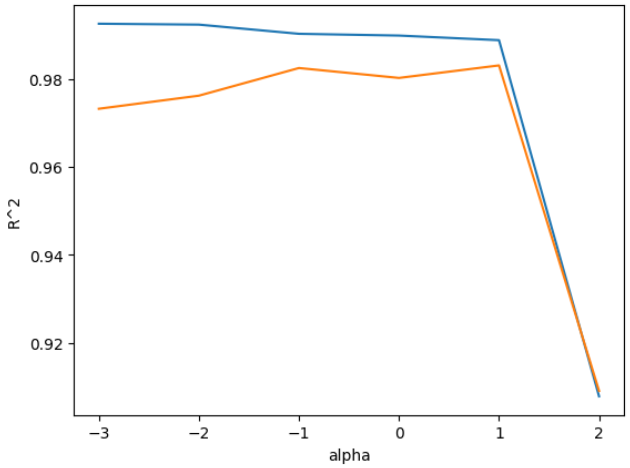


두 그래프가 가장 가깝고 테스트 세트 (주황) 의 점수가 가장 높은 -1. 즉, alpha 는 10의 -1 승인 0.1

2. 라쏘

- 많은 변수 중에서 중요한 변수만 선택하고 싶을 때

```
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))

train_score = []
test_score = []

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



라쏘 모델의 최적의 alpha 값은 10의 1승, 10.