Undergraduate Research Internship in Affective AI LAB.

혼자 공부하는 머신러닝 & 딥러닝

2주차 : ch03. 회귀 알고리즘과 모델 규제

컴퓨터공학과 12201682 공지윤

목차

기존 인공지능 모델의 문제와 해결 ①

피처 : perch_length / 라벨 : perch_weight

In [7]:

```
1 from sklearn.neighbors import KNeighborsRegressor
2 knr = KNeighborsRegressor()
3 knr.fit(train_input, train_target)
4 print("Test score :\t", knr.score(test_input, test_target))
5 print("Train score :\t", knr.score(train_input, train_target))
```

```
Test score :    0.992809406101064
Train score :   0.9698823289099254
```

Test > Train

- 분류 score : accuracy, precision, F1 score, recall 등
- 회귀 score : 결정계수 (R^2), MAE, MSE 등

기존 인공지능 모델의 문제와 해결 ①

결정계수 (R^2)

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

↗ 모델로 설명 불가능한 에러

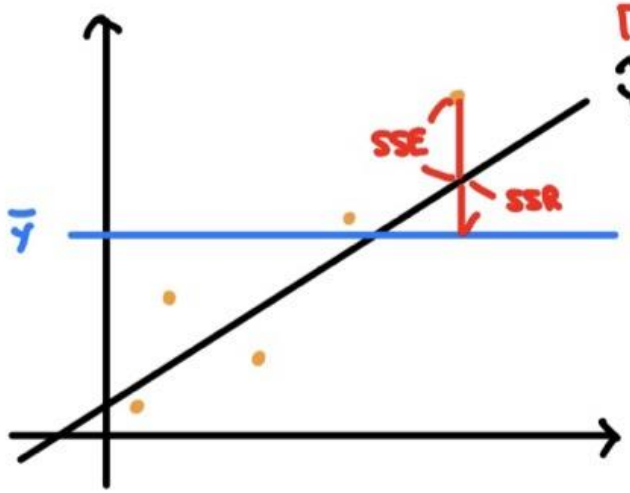
$$SST = SSE + SSR$$

Total Error sum Regression sum

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

추정량 (∧ hat)

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$



∴ R^2 이 1에 가깝다 ⇒ SSE 가 0에 가깝다 ⇒ 모델로 설명 불가능한 에러가 작다 ⇒ 설명력이 높다!

기존 인공지능 모델의 문제와 해결 ①

문제 : K-최근접 이웃 회귀 모델에서 과소적합이 발생했다.

과소적합 (Underfitting) : 모델이 데이터를 충분히 반영하지 못하는 문제

- 원인 : 데이터의 개수가 적어서 / 모델이 너무 단순해서
- 해결 : 데이터의 개수를 늘린다! / 모델을 복잡하게 만든다!

해결 : K-최근접 이웃 회귀 모델의 이웃의 개수 K를 줄인다.

기존 인공지능 모델의 문제와 해결 ①

해결 : K-최근접 이웃 회귀 모델의 이웃의 개수 K를 줄인다.

```
Init signature:
KNeighborsRegressor(
    n_neighbors=5,
    *,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    metric_params=None,
```

```
In [9]:
1 knr.n_neighbors = 3
2 knr.fit(train_input, train_target)
3 print("Test score :\t", knr.score(test_input, test_target))
4 print("Train score :\t", knr.score(train_input, train_target))
```

```
Test score :    0.9746459963987609
Train score :   0.9804899950518966
```

test < train

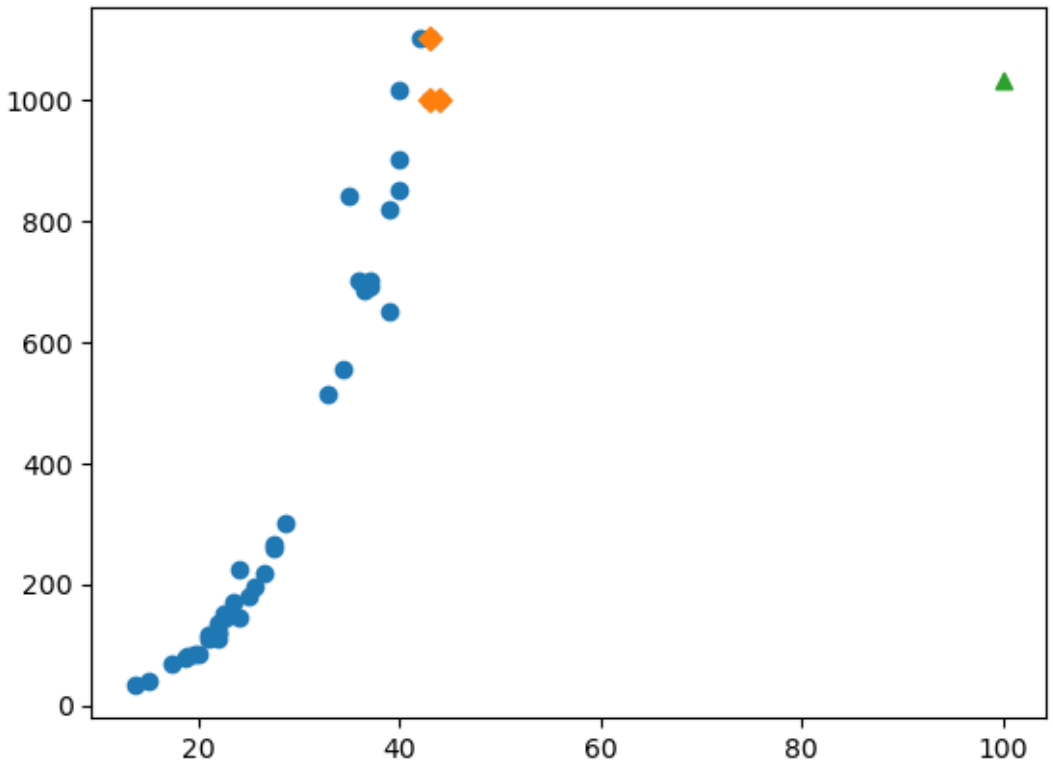
기존 인공지능 모델의 문제와 해결 ②

In [10]:
```
1 min(perch_length), max(perch_length)
```
Out[10]: (8.4, 44.0)

In [9]:
```
1 knr.predict([[50.0]]) # 실제 타깃값은 1500! 차이가 크다.
```
Out[9]: array([1033.33333333])

In [11]:
```
1 distances, indexes = knr.kneighbors([[50]])
2 train_target[indexes]
```
Out[11]: array([[1000., 1000., 1100.]])

In [12]:
```
1 knr.predict([[100.0]])
```
Out[12]: array([1033.33333333])



기존 인공지능 모델의 문제와 해결 ②

문제 : 훈련 세트의 범위를 벗어난 새로운 값에 대한 예측이 제대로 이뤄지지 않음

↓

해결 : K-최근접 이웃 말고 다른 모델을 사용한다.

[선형회귀]

In [11]:
```
1 from sklearn.linear_model import LinearRegression
2 lr = LinearRegression()
3 lr.fit(train_input, train_target)
4 lr.predict([[50.0], [100.0]])
```

Out[11]:
```
array([1241.83860323, 3192.69585141])
```

In [12]:
```
1 print(lr.coef_, lr.intercept_) # y = 39x - 709
```

```
[39.01714496] -709.0186449535477
```

In [13]:
```
1 print("Train score :\t", lr.score(train_input, train_target))
2 print("Test score :\t", lr.score(test_input, test_target)) # 과소적합 : test score가 높지 않음.
```

```
Train score :   0.939846333997604
Test score :    0.8247503123313558
```

기존 인공지능 모델의 문제와 해결 ②

문제 : 훈련 세트의 범위를 벗어난 새로운 값에 대한 예측이 제대로 이뤄지지 않음

↓

해결 : K-최근접 이웃 말고 다른 모델을 사용한다.

[다항회귀]

In [14]:
```
train_poly = np.column_stack((train_input ** 2, train_input))
test_poly = np.column_stack((test_input ** 2, test_input))
```

In [15]:
```
lr = LinearRegression()
lr.fit(train_poly, train_target)
print("Train score :\t", lr.score(train_poly, train_target))
print("Test score :\t", lr.score(test_poly, test_target)) # 과소적합
```

```
Train score :	 0.9706807451768623
Test score :	 0.9775935108325121
```

In [16]:
```
lr.predict([[50 ** 2, 50]])
```

Out[16]:
```
array([1573.98423528])
```

In [17]:
```
print(lr.coef_, lr.intercept_)
```

```
[  1.01433211 -21.55792498] 116.05021078278259
```

기존 인공지능 모델의 문제와 해결 ③

문제 : 다항회귀 모델에서 과소적합 문제가 발생했다.

↓

해결 : 더 복잡한 모델을 사용한다.

피처 : perch_length, perch_width, perch_height / 라벨 : perch_weight

In [27]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures()
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)
```

In [28]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(train_poly, train_target)
print("Train score :\t", lr.score(train_poly, train_target))
print("Test score :\t", lr.score(test_poly, test_target)) # 과소적합
```

```
Train score :    0.9903183436982125
Test score :     0.9714559911594094
```

[다중회귀]

기본 degree = 2

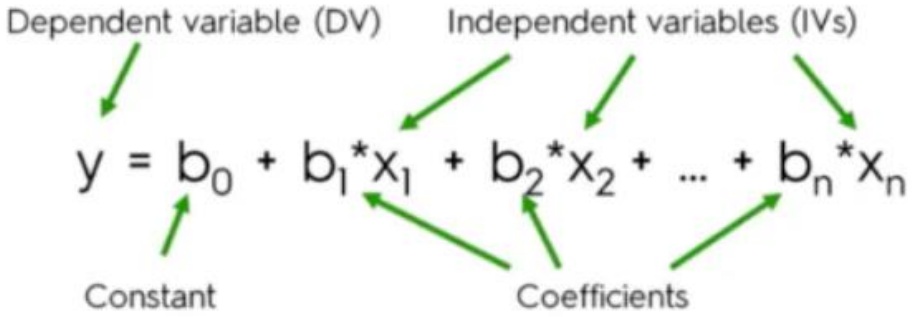
['1', 'x0', 'x1', 'x2', 'x0^2', 'x0 x1', 'x0 x2', 'x1^2', 'x1 x2', 'x2^2']

기존 인공지능 모델의 문제와 해결 ③

다중회귀 vs. 다항회귀

(1) 다중회귀 (Multiple Regression)

- 다중의 독립변수 존재
- 독립변수 간의 다중공선성 문제 처리 필요



(2) 다항회귀 (Polynomial Regression)

- 다차원 다항식으로 두고 회귀 분석 수행
- 다항회귀는 다중회귀로 계산될 수 있음

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$

기존 인공지능 모델의 문제와 해결 ④

In [30]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 5, include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)
```

In [31]:
```
lr.fit(train_poly, train_target)
print("Train score :\t", lr.score(train_poly, train_target))
print("Test score :\t", lr.score(test_poly, test_target)) # 과대적합
```

```
Train score :   0.9999999999938143
Test score :    -144.40744532797535
```

과대적합 (Overfitting) : 모델이 학습 데이터에 지나치게 최적화하여 발생하는 문제

- 원인 : 데이터의 개수가 적어서 / 데이터 내의 분산이 크거나 노이즈가 심해서 / 모델이 너무 복잡해서
  - 해결 : 데이터의 개수를 늘린다! / 데이터를 전처리 한다! / 모델의 복잡도를 낮춘다!

기존 인공지능 모델의 문제와 해결 ④

문제 : 다항회귀 모델에서 과대적합 문제가 발생했다.

↓

해결 : 규제 모델을 사용한다.

(1) Ridge Regression (릿지 회귀, L2 Regression)

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

- 잔차제곱합 (RSS) + 패널티 항
- 패널티 항 : 파라미터의 제곱의 합
- 패널티 항의 미분을 통한 최적화 가능
- 어떠한 파라미터도 0으로 만들지 않음
- λ : 패널티 부여 정도

In [34]:

```
1 from sklearn.linear_model import Ridge
2 ridge = Ridge(alpha=0.1)
3 ridge.fit(train_scaled, train_target)
4 print("Train score :\t", ridge.score(train_scaled, train_target))
5 print("Test score :\t", ridge.score(test_scaled, test_target))
```

```
Train score :    0.9903815817570368
Test score :     0.9827976465386896
```

기존 인공지능 모델의 문제와 해결 ④

(2) Lasso Regression (라쏘 회귀, L1 Regression)

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

- 잔차제곱합 (RSS) + 패널티 항
- 패널티 항 : 파라미터의 절대값의 합
- 파라미터를 0으로 만들어서 해당 변수 삭제 가능 (변수선택 기능)
- 살아남은 파라미터의 독립변수가 설명력이 좋아야 우수한 성능을 기대할 수 있음

기존 인공지능 모델의 문제와 해결 ④

(2) Lasso Regression (라쏘 회귀, L1 Regression)

In [35]:
```
1 from sklearn.linear_model import Lasso
2 lasso = Lasso(alpha=10)
3 lasso.fit(train_scaled, train_target)
4 print("Train score :\t", lasso.score(train_scaled, train_target))
5 print("Test score :\t", lasso.score(test_scaled, test_target))
```

```
Train score :    0.9888067471131867
Test score :     0.9824470598706695
```

In [55]:
```
1 len(lasso.coef_)
```

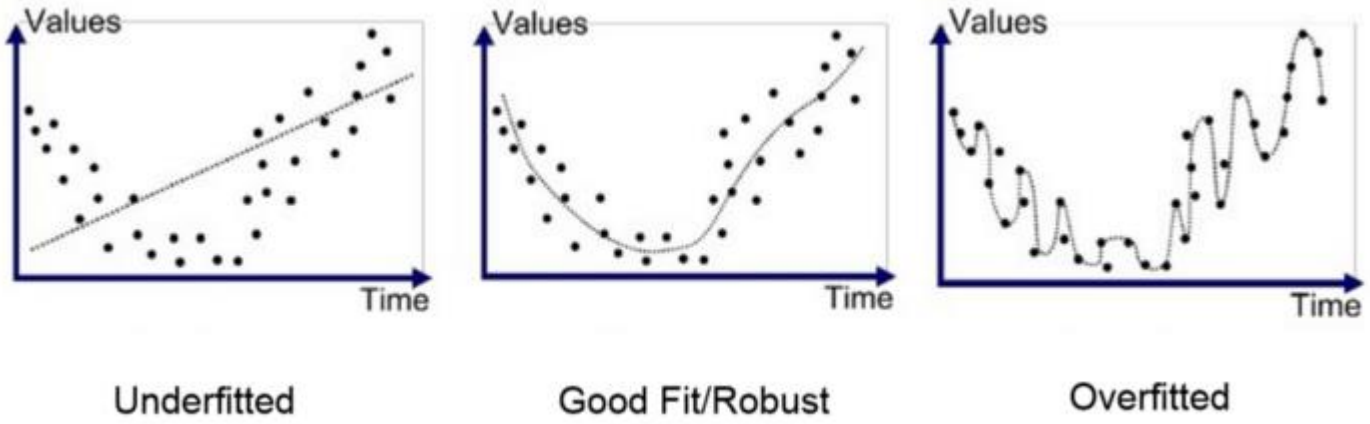
Out[55]: 55

In [56]:
```
1 np.sum(lasso.coef_ == 0)
```

Out[56]: 40

Summary

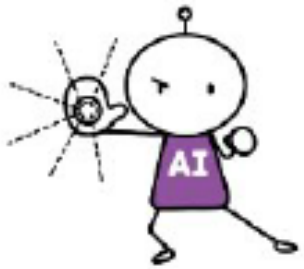
1. 결정계수는 회귀 모델의 설명력을 평가하는 지표이다.
2. 과대적합 vs. 과소적합



3. 과대적합이 발생했을 때 릿지 회귀나 라쏘 회귀로 모델의 복잡도를 낮춘다.

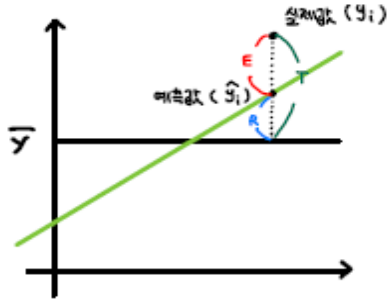
Undergraduate Research Internship in Affective AI LAB.

감사합니다 :>



컴퓨터공학과 12201682 공지윤

별첨) SST = SSR + SSE 식이 성립하는 이유 증명



$E = y_i - \hat{y}_i$ (실제값과 예측값의 차이) 회귀식으로 설명 불가능 (오차)

$R = \hat{y}_i - \bar{y}$ (예측값과 평균의 차이) 회귀식으로 설명 가능 (x 값에 따라 y값이 기울기 따라 얼마큼 변화가 예측됨에 따라 가능)

$T = y_i - \bar{y}$ (실제값과 평균의 차이)

$$T = R + E$$

$$(y_i - \bar{y}) = (\hat{y}_i - \bar{y}) + (y_i - \hat{y}_i)$$

오차 제곱 → 오차의 합이 0이 되기 때문에 제곱 처리해주기!

$SST = SSR + SSE$ (단, $\hat{y}_i$이 최소제곱법으로 구한 선형회귀모델의 y_i의 예측값일 때)

Proof) SST - SSR - SSE

$= \sum (y_i - \bar{y})^2 - \sum (y_i - \hat{y}_i)^2 - \sum (\hat{y}_i - \bar{y})^2$

$= \sum (y_i - \bar{y})^2 - (y_i - \hat{y}_i)^2 - (\hat{y}_i - \bar{y})^2$

$= \sum [(y_i^2 + \bar{y}^2 - 2y_i\bar{y}) - (y_i^2 - 2y_i\hat{y}_i + \hat{y}_i^2) - (\hat{y}_i^2 - 2\hat{y}_i\bar{y} + \bar{y}^2)]$

$= \sum (-2\hat{y}_i^2 - 2y_i\bar{y} + 2y_i\hat{y}_i + 2\hat{y}_i\bar{y})$ $\xrightarrow{\div(-2)}$ $\sum (\hat{y}_i^2 + y_i\bar{y} - y_i\hat{y}_i - \hat{y}_i\bar{y})$

$= \sum (\hat{y}_i^2 - y_i\hat{y}_i + y_i\bar{y} - \hat{y}_i\bar{y})$

$= \sum [\hat{y}_i(\hat{y}_i - y_i) + \bar{y}(y_i - \hat{y}_i)]$

$= \sum \hat{y}_i(\hat{y}_i - y_i) + \sum \bar{y}(y_i - \hat{y}_i)$

$= \sum \hat{y}_i(\hat{y}_i - y_i) + \bar{y} \sum (y_i - \hat{y}_i)$

↳ $\frac{\sum y_i}{n} = \frac{\sum \hat{y}_i}{n}$ (실제값의 평균 = 예측값의 평균)

$\hat{y}_i - y_i = e_i$

$\sum \hat{y}_i e_i$

$\hat{y}_i = \beta_0 + \beta_1 x_i$ ∴ 0

$\sum \hat{y}_i e_i = \sum (\beta_0 + \beta_1 x_i) e_i$

$= \sum (\beta_0 e_i + \beta_1 x_i e_i)$

$= \beta_0 \sum e_i + \beta_1 \sum x_i e_i$ ($\sum e_i$ = 0)

$= \beta_1 \sum x_i e_i = \beta_1 \sum x_i (\hat{y}_i - y_i)$

$= \beta_1 \sum x_i (\beta_0 + \beta_1 x_i - y_i)$

$= \beta_1 \sum \beta_0 x_i + \beta_1 x_i^2 - x_i y_i$

$= -\beta_1 \sum (x_i y_i - \beta_0 x_i - \beta_1 x_i^2)$

∴ 0

최소제곱법 → $\sum (y_i - \hat{y}_i)^2$이 최소가 되는 β_0과 β_1 찾기

$\frac{\partial}{\partial \beta_0} \sum (y_i - (\beta_1 x_i + \beta_0))^2$

$= \frac{\partial}{\partial \beta_0} \sum (y_i^2 - 2y_i(\beta_1 x_i + \beta_0) + (\beta_1 x_i + \beta_0)^2)$

$= \frac{\partial}{\partial \beta_0} \sum (y_i^2 - 2y_i\beta_1 x_i - 2\beta_0 y_i + (\beta_1 x_i + \beta_0)^2)$

$= \sum (-2y_i + 2(\beta_1 x_i + \beta_0)) = 0$

$\therefore \sum y_i = \sum \beta_1 x_i + \beta_0$

$= \sum \hat{y}_i$

$\frac{\partial}{\partial \beta_1} \sum (y_i - (\beta_1 x_i + \beta_0))^2$

$= \frac{\partial}{\partial \beta_1} \sum (y_i^2 - 2y_i\beta_1 x_i - 2\beta_0 y_i + (\beta_1 x_i + \beta_0)^2)$

$= \sum [-2y_i x_i + 2(\beta_1 x_i + \beta_0) x_i]$

$= -2 \sum [x_i y_i - (\beta_1 x_i + \beta_0) x_i]$

$= -2 \sum (x_i y_i - \beta_1 x_i^2 - \beta_0 x_i) = 0$

$\sum (x_i y_i - \beta_1 x_i^2 - \beta_0 x_i) = 0$