05_머신러닝 회귀

머신러닝 회귀

• 회귀란?

> 여러 개의 독립 변수와 한 개의 종속변수 간의 상관관계 모델링

> $\boldsymbol{Y = W_0X_0 + W_1X_1 + W_2X_2 + \cdots + W_NX_N}$ 일 경우

- Y : 종속변수, 레이블
- X : 독립변수, 피쳐
- W : 회귀 계수

> 최적의 회귀 계수를 찾아내는 일이 회귀 분석이다.

> 독립 변수가 1개면 단일 회귀, 다수면 다중 회귀로 나뉜다.

> 회귀 계수의 결합에 따라 선형이냐 비선형이냐로 나뉜다.

머신러닝 회귀

• 회귀란?

> 선형 결합이란?

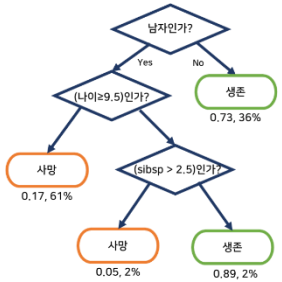
- 회귀 계수를 선형 모델로 만들 수 있는 상태

  - $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 \Rightarrow \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$
  - $y = \beta_0 x^{\beta_1} \Rightarrow log(y) = log(\beta_0 x^{\beta_1}) \Rightarrow log\beta_0 + \beta_1 \log(x) \Rightarrow y^* = \beta_0^* + \beta_1 x^*$
  - $y = \frac{e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}}{1 + e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}} \Rightarrow \frac{y}{1-y} = e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3} \Rightarrow \log\left(\frac{y}{1-y}\right) = y^* = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$

> 비선형 결합이란?

- 선형 모델로 표현이 불가능한 상태

$$y = \frac{\beta_1 x}{\beta_2 + x}$$



머신러닝 회귀

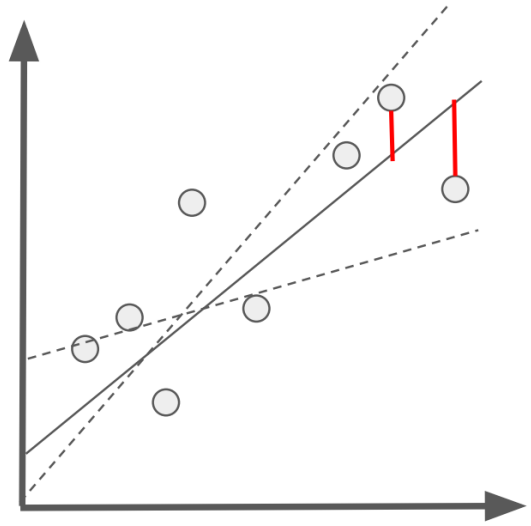
• 회귀란?

> 대표적인 선형 회귀

- 일반 선형 회귀 : 예측값과 실제값의 오류를 최소화할 수 있도록 회귀 계수를 최적화화며, 규제(Regularization)를 적용하지 않는 모델
- 릿지(Ridge) : 선형 회귀에 L2 규제를 추가한 모델
- 라쏘(Lasso) : 선형 회귀에 L1 규제를 추가한 모델
- 엘리스틱넷 : L2, L1 규제를 결합한 모델
- 로지스틱 회귀 : 회귀보단 이진 분류에 특화된 모델
- SGD : 확률적 경사 하강법을 적용한 선형 모델

• 단순 선형 회귀

> 독립변수 하나, 종속변수 하나인 선형 회귀
  - $Y = W_0 + W_1 * x$
> 오류 : 실제 값과 회귀 모델의 차이
  - 오류의 합을 최소화 하는 회귀 선을 구한다.
> RSS : 오류 값의 제곱을 더해 평균을 내는 방식
> MAE : 오류 값의 절대값을 더해 평균을 내는 방식



$$RSS(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w_0 + w_1 * x_i))^2$$

• 선형 회귀 분석

> 당뇨병 수치 예측

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from scipy import stats
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()

diabetes_df = pd.DataFrame(diabetes.data , columns = diabetes.feature_names)
diabetes_df['target'] = diabetes.target
```

• 선형 회귀 분석

> 당뇨병 수치 예측

```
diabetes_df.head()
```

Out :

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0

• 선형 회귀 분석

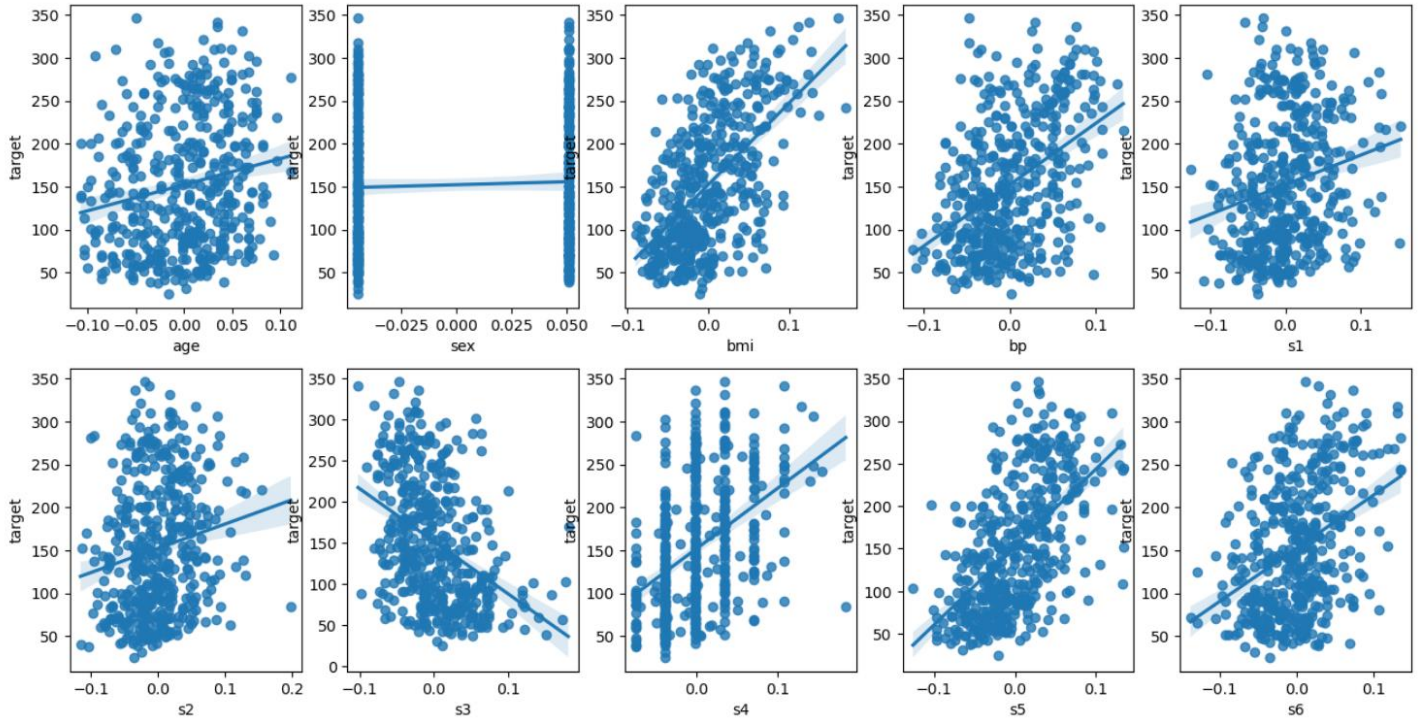
> 당뇨병 수치 예측

```
fig, axs = plt.subplots(figsize=(16,8) , ncols=5 , nrows=2)
lm_features = ['age','sex','bmi','bp','s1','s2','s3','s4', 's5', 's6']

for i , feature in enumerate(lm_features):
    row = int(i/5)
    col = i%5
    sns.regplot(x=feature , y='target',data=diabetes_df, ax=axs[row][col])
diabetes_df.corr()['target']
```

• 선형 회귀 분석

> 당뇨병 수치 예측

```
age        0.187889
sex        0.043062
bmi        0.586450
bp         0.441482
s1         0.212022
s2         0.174054
s3        -0.394789
s4         0.430453
s5         0.565883
s6         0.382483
target     1.000000
Name: target, dtype: float64
```



• 선형 회귀 분석

> 당뇨병 수치 예측

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


y = diabetes_df.iloc[:, -1]
X = diabetes_df.iloc[:, :-1]
X_train , X_test , y_train , y_test = train_test_split(X, y,
                                                       test_size=0.3, random_state=156)
```

• 선형 회귀 분석

> 당뇨병 수치 예측

```
lr = LinearRegression()
lr.fit(X_train ,y_train )
pred = lr.predict(X_test)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
print(f'MSE : {mse:.3f} , RMSE : {rmse:.3F}')
print(f'Variance score : {r2_score(y_test, pred):.3f}')
```

Out : MSE : 2993.705 , RMSE : 54.715

Variance score : 0.497

• 선형 회귀 분석

> 당뇨병 수치 예측

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

평가 지표	설명	수식
MAE	Mean Absolute Error(MAE)이며 실제 값과 예측값의 차이를 절댓값으로 변환해 평균한 것입니다	$MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert Yi - \hat{Y}i\rvert$
MSE	Mean Squared Error(MSE)이며 실제 값과 예측값의 차이를 제곱해 평균한 것입니다.	$MSE = \frac{1}{n}\sum_{i=1}^{n}(Yi - \hat{Y}i)^2$
MSLE	MSE에 로그를 적용한 것입니다. 결정값이 클 수록 오류값도 커지기 때문에 일부 큰 오류값들로 인해 전체 오류값이 커지는 것을 막아줍니다.	Log(MSE)
RMSE	MSE 값은 오류의 제곱을 구하므로 실제 오류 평균보다 더 커지는 특성이 있으므로 MSE에 루트를 씌운 것이 RMSE(Root Mean Squared Error)입니다	$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Yi - \hat{Y}i)^2}$
RMSLE	RMSE에 로그를 적용한 것입니다. 결정값이 클 수록 오류값도 커지기 때문에 일부 큰 오류값들로 인해 전체 오류값이 커지는 것을 막아줍니다.	Log(RMSE)
R^2	분산 기반으로 예측 성능을 평가합니다. 실제 값의 분산 대비 예측값의 분산 비율을 지표로 하며, 1에 가까울수록 예측 정확도가 높습니다.	$R^2 = \frac{\text{예측값 } Variance}{\text{실제 값 } Variance}$

• 선형 회귀 분석

> 당뇨병 수치 예측

```
print('절편 값:',lr.intercept_)
print('회귀 계수값:', np.round(lr.coef_, 1))
```

Out : 절편 값: 152.38617209733573
회귀 계수값: [39.1 -251.9 468.8 305.3 -1146.9 788. 177.2 117.4
937.9 53.7]

```
coeff = pd.Series(data=np.round(lr.coef_, 1), index=X.columns )
coeff
```

Out :

```
age      39.1
sex    -251.9
bmi     468.8
bp      305.3
s1    -1146.9
s2      788.0
s3      177.2
s4      117.4
s5      937.9
s6       53.7
dtype: float64
```

• 선형 회귀 분석

> 당뇨병 수치 예측

```
from sklearn.model_selection import cross_val_score
lr = LinearRegression()

neg_mse_scores = cross_val_score(lr, X, y, scoring="neg_mean_squared_error", cv = 5)
# 'neg_root_mean_squared_error', 'neg_mean_absolute_error', 'r2'

rmse_scores  = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print('5 folds 의 개별 Negative MSE scores:', np.round(neg_mse_scores, 2))
print('5 folds 의 개별 RMSE scores :', np.round(rmse_scores, 2))
print(f'5 folds 의 평균 RMSE : {avg_rmse:.3f}')
```

• 선형 회귀 분석

> 당뇨병 수치 예측

Out :
```
5 folds 의 개별 Negative MSE scores: [-2779.92 -3028.84 -3237.69 -3008.75 -2910.21]
5 folds 의 개별 RMSE scores : [52.72 55.03 56.9  54.85 53.95]
5 folds 의 평균 RMSE : 54.692
```

• 다항회귀

> 다항회귀 실습

```
from sklearn.preprocessing import PolynomialFeatures
import numpy as np

X = np.arange(4).reshape(2,2)
print('일차 단항식 계수 feature:₩n',X )
poly = PolynomialFeatures(degree=2)
poly.fit(X)
poly_ftr = poly.transform(X)
print('변환된 2차 다항식 계수 feature:₩n', poly_ftr)
```

- 다항회귀

> 다항회귀 실습

x_1	x_2
0	1
2	3

1차 식의 데이터를
2차 식의 데이터 변환하는 방식
* bias라는 조건이 붙는다

bias	x_1	x_2	x_1^2	x_1x_2	x_2^2
1	0	1	0	0	1
1	2	3	4	6	9

• 다항회귀

> 다항회귀 실습

```
def polynomial_func(X):
    y = 1 + 2*X[:,0] + 3*X[:,0]**2 + 4*X[:,1]**3
    return y

X = np.arange(0,4).reshape(2,2)
print('일차 단항식 계수 feature: ₩n' ,X)
y = polynomial_func(X)
print('삼차 다항식 결정값: ₩n', y)
```

Out :

```
일차 단항식 계수 feature:
 [[0 1]
 [2 3]]
삼차 다항식 결정값:
 [  5 125]
```

• 다항회귀

> 다항회귀 실습

```
poly_ftr = PolynomialFeatures(degree=3).fit_transform(X)
print('3차 다항식 계수 feature: ₩n',poly_ftr)
```

Out : 3차 다항식 계수 feature:
[[1. 0. 1. 0. 0. 1. 0. 0. 0. 1.]
[1. 2. 3. 4. 6. 9. 8. 12. 18. 27.]]

> 다음과 같은 규칙을 가지고 생성한다.

x_1	x_2
0	1
2	3

→

bias	x_1	x_2	x_1^2	x_1x_2	x_2^2	x_1^3	$x_1^2x_2$	$x_1x_2^2$	x_2^3
1	0	1	0	0	1	0	0	0	1
1	2	3	4	6	9	8	12	18	27

• 다항회귀

> 다항회귀 실습

```
from sklearn.linear_model import LinearRegression


model = LinearRegression()
model.fit(poly_ftr,y)
print('Polynomial 회귀 계수₩n' , np.round(model.coef_, 2))
print('Polynomial 회귀 Shape :', model.coef_.shape)
```

Out : Polynomial 회귀 계수
[0. 0.18 0.18 0.36 0.54 0.72 0.72 1.08 1.62 2.34]
Polynomial 회귀 Shape : (10,)

• 다항회귀와 과적합/과소적합

> 과적합, 과소적합 이해

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

def true_fun(X):
    return np.cos(1.5 * np.pi * X)
np.random.seed(0)
n_samples = 30
X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1
```

• 다항회귀와 과적합/과소적합

> 과적합, 과소적합 이해

```
plt.figure(figsize=(14, 5))
degrees = [1, 4, 15]

for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    lr_reg = LinearRegression()
    X_poly = polynomial_features.fit_transform(X.reshape(-1,1), y)
    lr_reg.fit(X_poly, y)
    scores = cross_val_score(lr_reg, X_poly, y, scoring='neg_mean_squared_error', cv=10)
    coeff = lr_reg.coef_
```

• 다항회귀와 과적합/과소적합

> **과적합, 과소적합 이해**

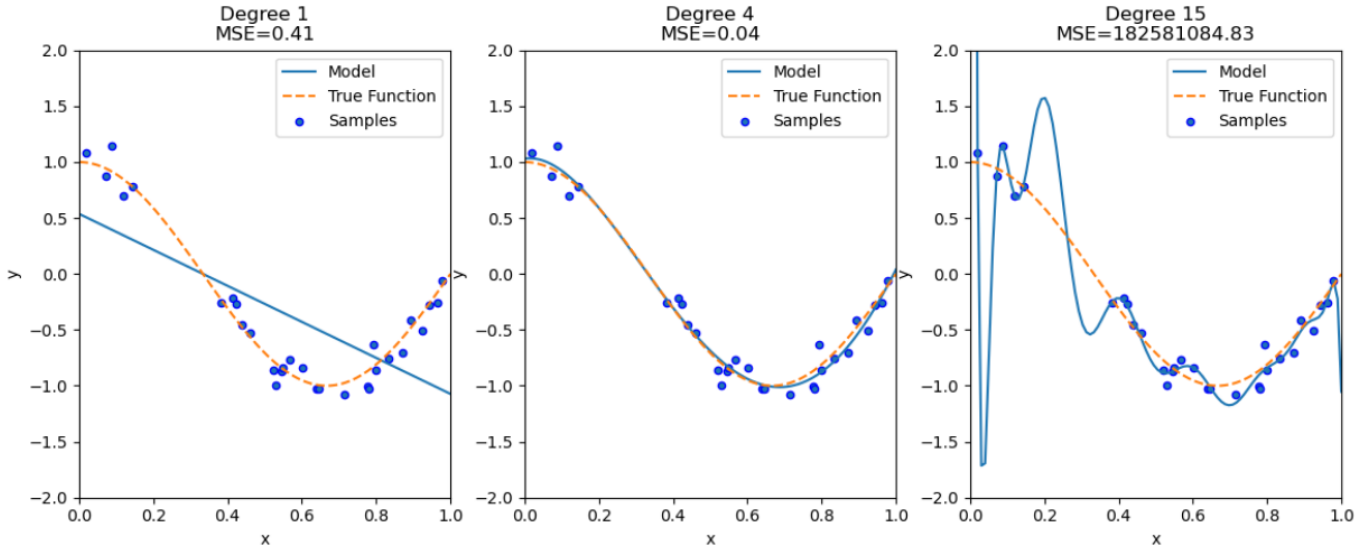
```
    # 이전 코드에 이어서 작성
    print(f'₩nDegree {degrees[i]} 회귀계수는 ₩n{np.round(coeff, 2)}₩n입니다.')
    print(f'MSE는 {-1 * np.mean(scores)}입니다.')
    X_test = np.linspace(0, 1, 100).reshape(-1, 1)
    X_test_poly = polynomial_features.transform(X_test)
    pred = lr_reg.predict(X_test_poly)
    plt.plot(X_test, pred, label = 'Model')
    plt.plot(X_test, true_fun(X_test), '--', label='True Function')
    plt.scatter(X, y, edgecolors='b', s = 20, label = 'Samples')
    plt.xlabel('x');plt.ylabel('y'); plt.xlim((0,1)); plt.ylim((-2, 2)); plt.legend()
    plt.title(f'Degree {degrees[i]}₩nMSE={-scores.mean():.2f}')
plt.show()
```

• 다항회귀와 과적합/과소적합

> 과적합, 과소적합 이해

Out :

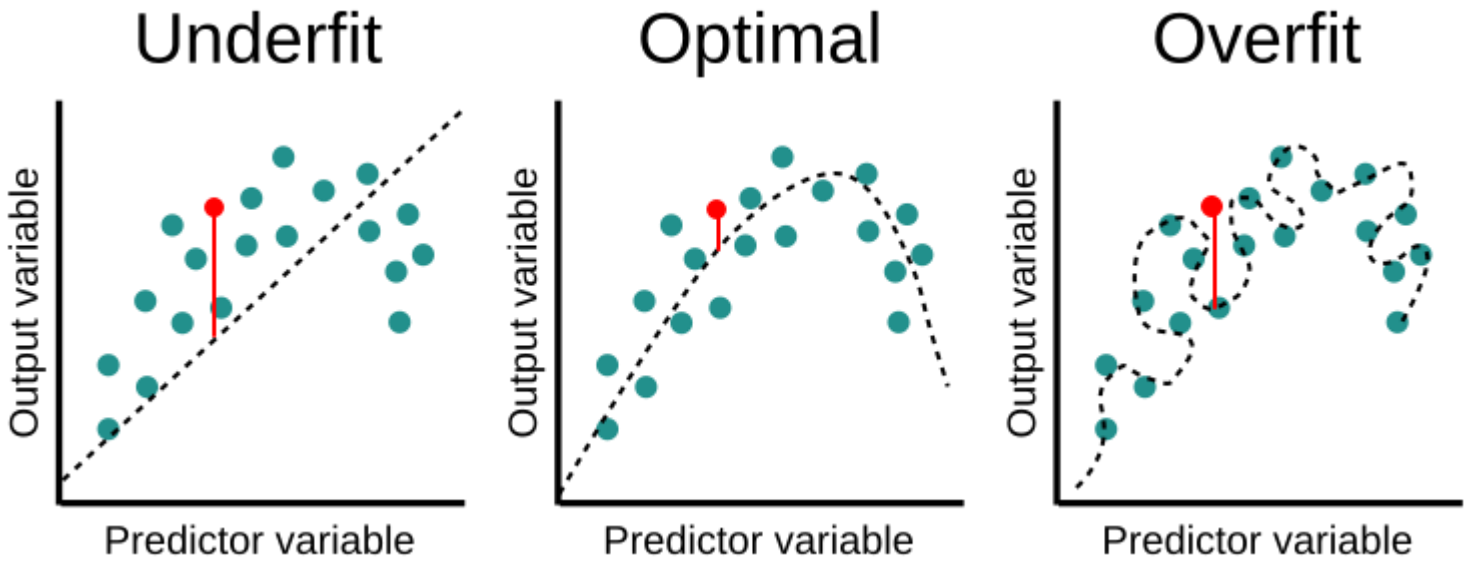
```
Degree 1 회귀계수는
[-1.61]
입니다.
MSE는 0.40772896250986845입니다.

Degree 4 회귀계수는
[  0.47 -17.79  23.59  -7.26]
입니다.
MSE는 0.04320874987231840입니다.

Degree 15 회귀계수는
[-2.98294000e+03  1.03899850e+05 -1.87416981e+06  2.03717199e+07
 -1.44874017e+08  7.09319141e+08 -2.47067173e+09  6.24564702e+09
 -1.15677216e+10  1.56895933e+10 -1.54007040e+10  1.06457993e+10
 -4.91381016e+09  1.35920643e+09 -1.70382078e+08]
입니다.
MSE는 182581084.8263125입니다.
```



• 과적합/과소적합

> **과적합**
  - 데이터를 학습할 때, 학습 데이터에만 과하게 학습을 하여 일반화하지 못하게 되어 테스트 데이터에서 오차가 커지는 현상

> **과소적합**
  - 학습 데이터에 대한 학습이 부족하여 훈련/테스트 데이터에서 오차가 크게 나오는 현상



• 규제 선형 모델

> 릿지 회귀 : L2 규제

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score

y = diabetes_df.iloc[:, -1]
X = diabetes_df.iloc[:, :-1]
ridge = Ridge(alpha = 10)
neg_mse_scores = cross_val_score(ridge, X, y,scoring="neg_mean_squared_error", cv = 5)
rmse_scores  = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)
print(' 5 folds 의 개별 Negative MSE scores: ', np.round(neg_mse_scores, 3))
print(' 5 folds 의 개별 RMSE scores : ', np.round(rmse_scores,3))
print(' 5 folds 의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

• 규제 선형 모델

> 릿지 회귀 : L2 규제

Out :
```
5 folds 의 개별 Negative MSE scores:  [-4540.477 -5430.505 -5284.404 -4364.15  -5463.355]
5 folds 의 개별 RMSE scores :  [67.383 73.692 72.694 66.062 73.915]
5 folds 의 평균 RMSE : 70.749
```

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

• 규제 선형 모델

> 릿지 회귀 : L2 규제

```
alphas = [0, 0.1, 1, 10, 100]
for alpha in alphas :
    ridge = Ridge(alpha = alpha)

    neg_mse_scores = cross_val_score(ridge, X, y,
                                     scoring="neg_mean_squared_error", cv = 5)
    avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
    print(f'alpha {alpha} 일 때 5 folds 의 평균 RMSE : {avg_rmse:.3f} ')
```

Out :

```
alpha 0 일 때 5 folds 의 평균 RMSE : 54.692
alpha 0.1 일 때 5 folds 의 평균 RMSE : 54.825
alpha 1 일 때 5 folds 의 평균 RMSE : 58.449
alpha 10 일 때 5 folds 의 평균 RMSE : 70.749
alpha 100 일 때 5 folds 의 평균 RMSE : 76.399
```

• 규제 선형 모델

> 릿지 회귀 : L2 규제

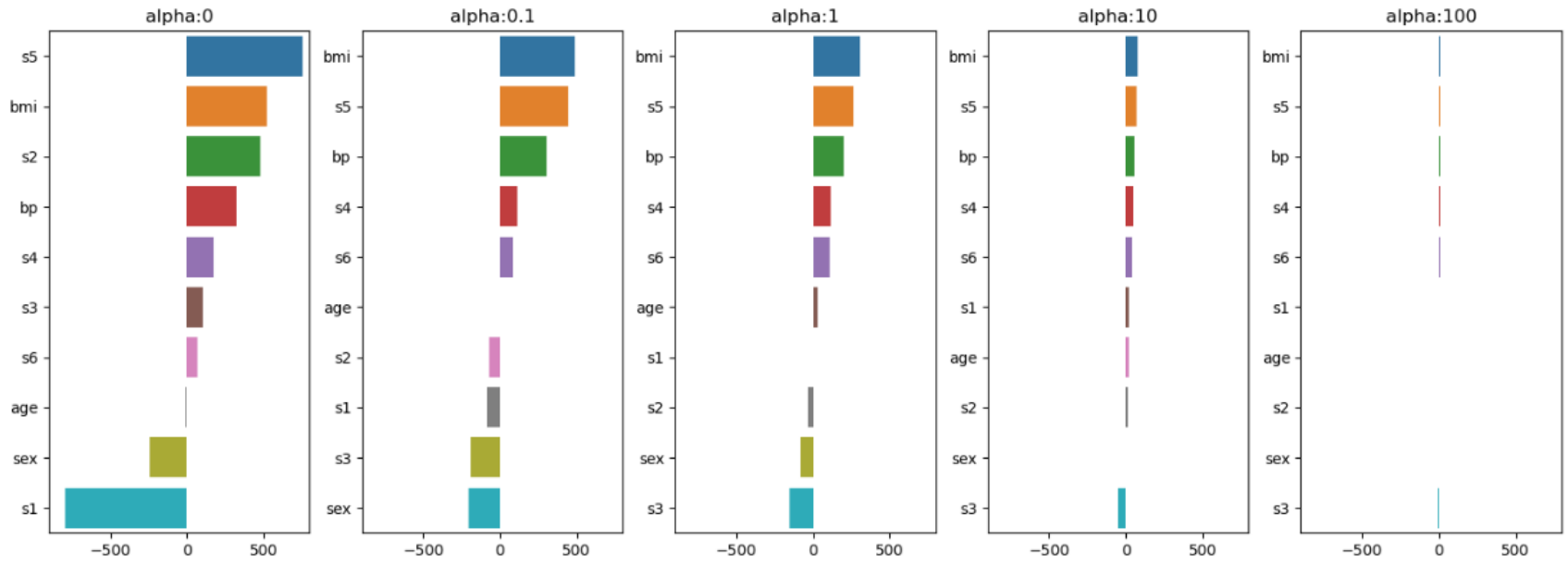
```
fig, axs = plt.subplots(figsize = (18,6), nrows=1, ncols=5)
coeff_df = pd.DataFrame()
for pos, alpha in enumerate(alphas):
    ridge = Ridge(alpha = alpha)
    ridge.fit(X, y)
    coeff = pd.Series(data = ridge.coef_, index = X.columns)
    colname='alpha:'+ str(alpha)
    coeff_df[colname] = coeff
    coeff = coeff.sort_values(ascending = False)
    axs[pos].set_title(colname); axs[pos].set_xlim(-900, 800)
    sns.barplot(x = coeff.values, y=coeff.index, ax=axs[pos])
plt.show()
```

머신러닝 회귀

- 규제 선형 모델

> 릿지 회귀 : L2 규제

Out :



• 규제 선형 모델

> 릿지 회귀 : L2 규제

```
ridge_alphas = [0 , 0.1 , 1 , 10 , 100]
sort_column = 'alpha:'+ str(ridge_alphas[0])
coeff_df.sort_values(by=sort_column, ascending=False)
```

Out :

	alpha:0	alpha:0.1	alpha:1	alpha:10	alpha:100
s5	751.273700	443.812917	262.944290	70.143948	8.876851
bmi	519.845920	489.695171	306.352680	75.416214	9.240720
s2	476.739021	-70.826832	-29.515495	13.948715	2.616766
bp	324.384646	301.764058	201.627734	55.025160	6.931289
s4	177.063238	115.712136	117.311732	48.259433	6.678027
s3	101.043268	-188.678898	-152.040280	-47.553816	-6.174550
s6	67.626692	86.749315	111.878956	44.213892	5.955597
age	-10.009866	1.308705	29.466112	19.812842	2.897090
sex	-239.815644	-207.192418	-83.154276	-0.918430	0.585254
s1	-792.175639	-83.466034	5.909614	19.924621	3.230957

규제 강도가
강할수록
회귀계수가
작아진다.

• 규제 선형 모델

> 라쏘 회귀 : L1 규제

```
from sklearn.linear_model import Lasso
alphas = [0, 0.01, 0.1, 1, 10]
for alpha in alphas :
    lasso = Lasso(alpha = alpha)
    neg_mse_scores = cross_val_score(lasso, X, y,
                                     scoring="neg_mean_squared_error", cv = 5)
    avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
    print(f'alpha {alpha} 일 때 5 folds 의 평균 RMSE : {avg_rmse:.3f} ')
```

Out :

```
alpha 0 일 때 5 folds 의 평균 RMSE : 54.692
alpha 0.01 일 때 5 folds 의 평균 RMSE : 54.756
alpha 0.1 일 때 5 folds 의 평균 RMSE : 54.843
alpha 1 일 때 5 folds 의 평균 RMSE : 62.009
alpha 10 일 때 5 folds 의 평균 RMSE : 77.264
```

• 규제 선형 모델

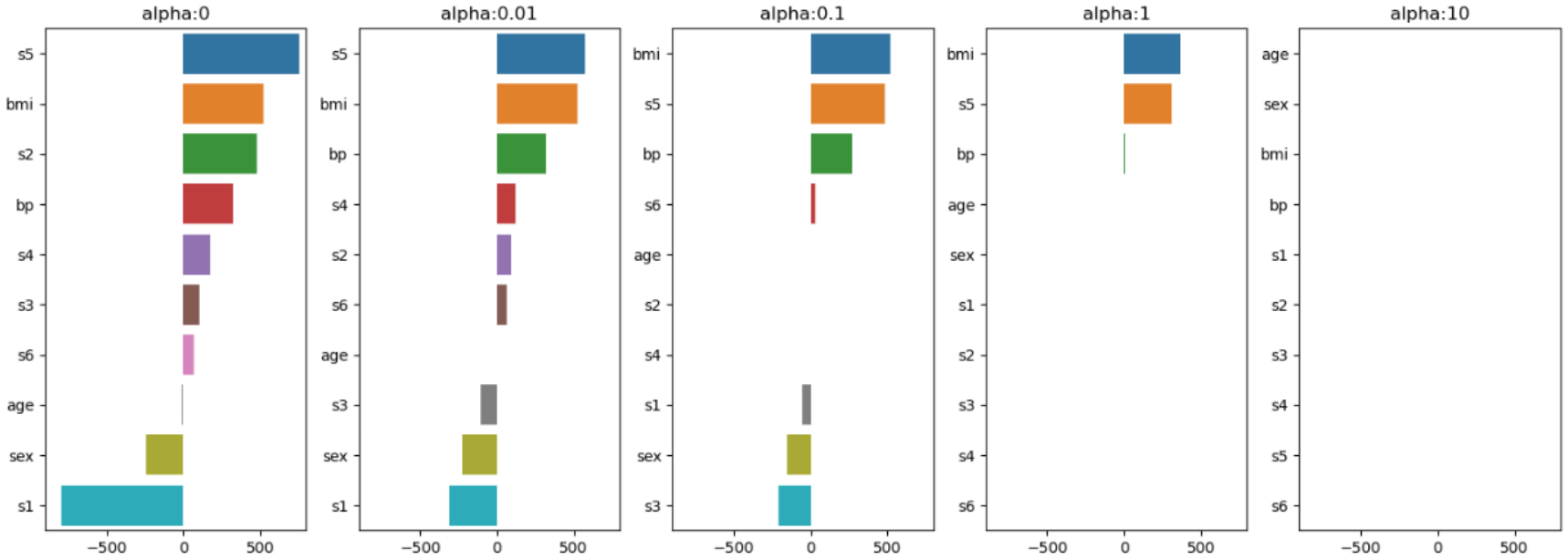
> 라쏘 회귀 : L1 규제

```
fig, axs = plt.subplots(figsize = (18,6), nrows=1, ncols=5)
coeff_df = pd.DataFrame()
for pos, alpha in enumerate(alphas):
    lasso = Lasso(alpha = alpha)
    lasso.fit(X, y)
    coeff = pd.Series(data = lasso.coef_, index = X.columns)
    colname='alpha:'+ str(alpha)
    coeff_df[colname] = coeff
    coeff = coeff.sort_values(ascending = False)
    axs[pos].set_title(colname); axs[pos].set_xlim(-900, 800)
    sns.barplot(x = coeff.values, y=coeff.index, ax=axs[pos])
plt.show()
```

- 규제 선형 모델

> 라쏘 회귀 : L1 규제

Out :



• 규제 선형 모델

> 라쏘 회귀 : L1 규제

```
lasso_alphas = [0 , 0.1 , 1 , 10 , 100]
sort_column = 'alpha:'+ str(lasso_alphas[0])
coeff_df.sort_values(by=sort_column, ascending=False)
```

Out :

	alpha:0	alpha:0.01	alpha:0.1	alpha:1	alpha:10
s5	751.273690	571.330356	483.912648	307.605418	0.0
bmi	519.845920	525.566130	517.186795	367.703860	0.0
s2	476.739000	89.324647	-0.000000	0.000000	0.0
bp	324.384645	316.168834	275.077235	6.298858	0.0
s4	177.063234	119.597616	0.000000	0.000000	0.0
s3	101.043256	-105.078369	-210.157991	-0.000000	-0.0
s6	67.626692	65.008383	33.673965	0.000000	0.0
age	-10.009866	-1.304662	-0.000000	0.000000	0.0
sex	-239.815644	-228.819129	-155.359976	-0.000000	0.0
s1	-792.175612	-307.016211	-52.539365	0.000000	0.0

L2 규제와 똑같이
규제 강도가 강할수록
회귀계수가 작아진다.

하지만 L1 규제는
강도가 강하면 0으로
수렴한다.

• 규제 선형 모델

> 엘라스틱넷 회귀 : L1, L2 규제

```
from sklearn.linear_model import ElasticNet
alphas = [0, 0.01, 0.1, 1, 10]
for alpha in alphas :
    elastic = ElasticNet(alpha = alpha)
    neg_mse_scores = cross_val_score(elastic, X, y,
                                     scoring="neg_mean_squared_error", cv = 5)
    avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
    print(f'alpha {alpha} 일 때 5 folds 의 평균 RMSE : {avg_rmse:.3f} ')
```

Out :

```
alpha 0 일 때 5 folds 의 평균 RMSE : 54.692
alpha 0.01 일 때 5 folds 의 평균 RMSE : 61.112
alpha 0.1 일 때 5 folds 의 평균 RMSE : 73.203
alpha 1 일 때 5 folds 의 평균 RMSE : 76.925
alpha 10 일 때 5 folds 의 평균 RMSE : 77.264
```

• 규제 선형 모델

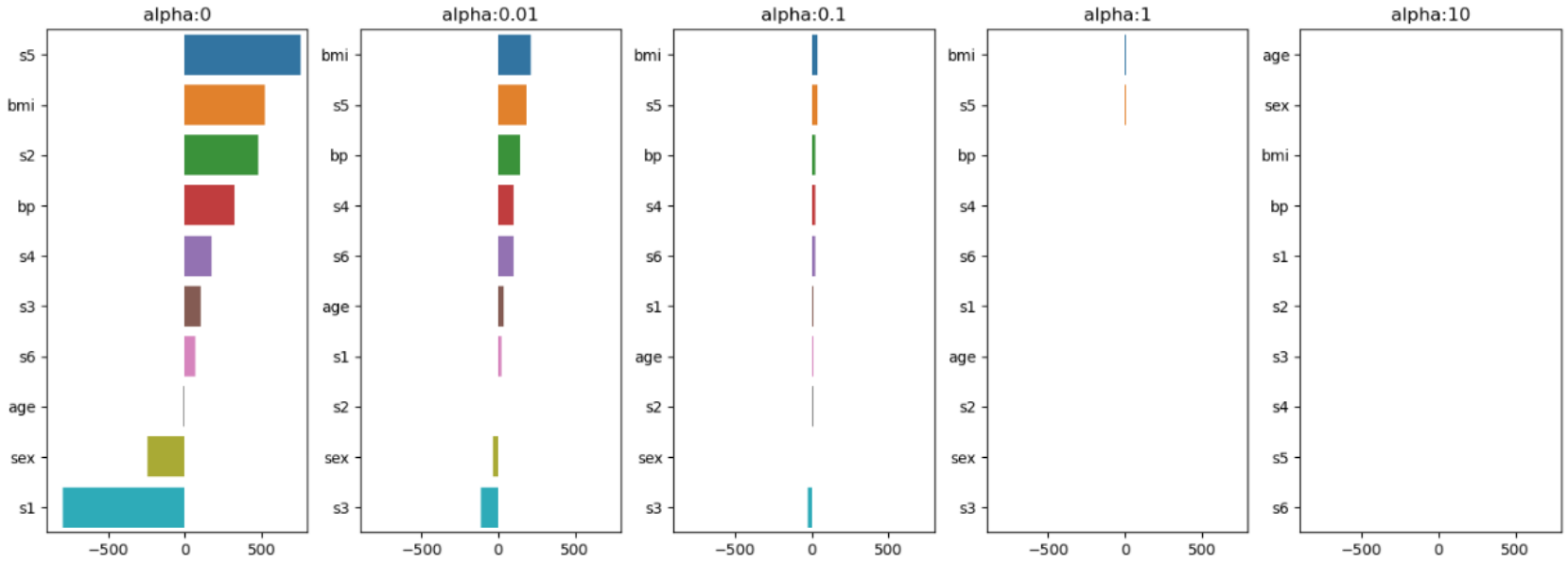
> 엘라스틱넷 회귀 : L1, L2 규제

```
fig, axs = plt.subplots(figsize = (18,6), nrows=1, ncols=5)
coeff_df = pd.DataFrame()
for pos, alpha in enumerate(alphas):
    elastic = ElasticNet(alpha = alpha)
    elastic.fit(X, y)
    coeff = pd.Series(data = elastic.coef_, index = X.columns)
    colname='alpha:'+ str(alpha)
    coeff_df[colname] = coeff
    coeff = coeff.sort_values(ascending = False)
    axs[pos].set_title(colname); axs[pos].set_xlim(-900, 800)
    sns.barplot(x = coeff.values, y=coeff.index, ax=axs[pos])
plt.show()
```

• 규제 선형 모델

> 엘라스틱넷 회귀 : L1, L2 규제

Out :



• 규제 선형 모델

> 엘라스틱넷 회귀 : L1, L2 규제

```
alphas = [0 , 0.1 , 1 , 10 , 100]
sort_column = 'alpha:'+ str(alphas[0])
coeff_df.sort_values(by=sort_column, ascending=False)
```

Out :

	alpha:0	alpha:0.01	alpha:0.1	alpha:1	alpha:10
s5	751.273690	185.325911	35.465700	3.105835	0.0
bmi	519.845920	211.024367	37.464655	3.259767	0.0
s2	476.739000	0.000000	8.355892	0.250935	0.0
bp	324.384645	144.559236	27.544765	2.204340	0.0
s4	177.063234	100.658917	25.505492	2.114454	0.0
s3	101.043256	-115.619973	-24.120809	-1.861363	-0.0
s6	67.626692	96.257335	22.894985	1.769851	0.0
age	-10.009866	33.147367	10.286332	0.359018	0.0
sex	-239.815644	-35.245354	0.285983	0.000000	0.0
s1	-792.175612	21.931722	11.108856	0.528646	0.0

마찬가지로 규제 강도가
클수록 회귀계수가
작아진다.

L1처럼 0으로
수렴하기도 하지만
L2처럼 천천히 값이
작아진다.

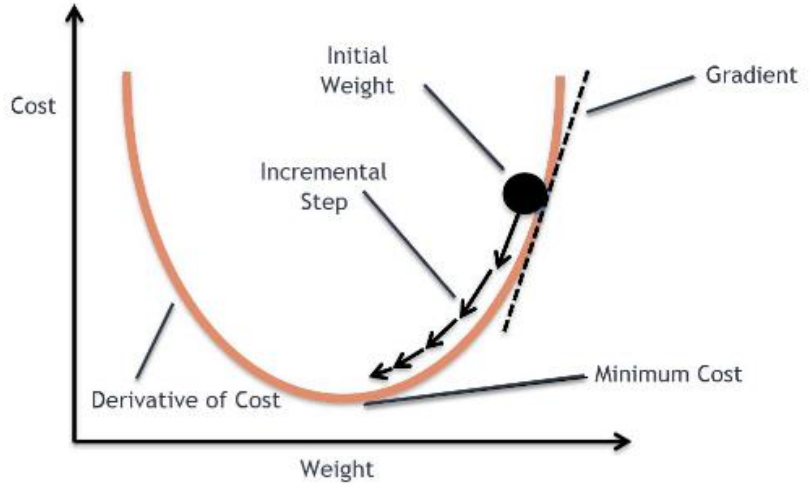
L1, L2 특징을 모두 표현

• 경사 하강법

> 비용(Cost)을 최소화 하는 방법
  - 미분 값이 0이 되는 지점이 최소점

$$\frac{\partial R(w)}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} -x_i * (y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$\frac{\partial R(w)}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} (\text{실제값}_i - \text{예측값}_i)$$



출처: https://towardsdatascience.com/using-machine-learning-to-predict-fitbit-sleep-scores-496a7d9ec48

• 경사 하강법

> 확률적 경사 하강법(SGD) 실습

```
from sklearn.linear_model import SGDRegressor
alphas = [0, 0.01, 0.1, 1, 10]
for alpha in alphas :
    sgd_reg = SGDRegressor(alpha=alpha)
    neg_mse_scores = cross_val_score(sgd_reg, X, y,
                                     scoring="neg_mean_squared_error", cv = 5)
    avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
    print(f'alpha {alpha} 일 때 5 folds 의 평균 RMSE : {avg_rmse:.3f} ')
```

Out :

```
alpha 0 일 때 5 folds 의 평균 RMSE : 59.020
alpha 0.01 일 때 5 folds 의 평균 RMSE : 65.432
alpha 0.1 일 때 5 folds 의 평균 RMSE : 74.984
alpha 1 일 때 5 folds 의 평균 RMSE : 76.999
alpha 10 일 때 5 folds 의 평균 RMSE : 77.244
```

• 경사 하강법

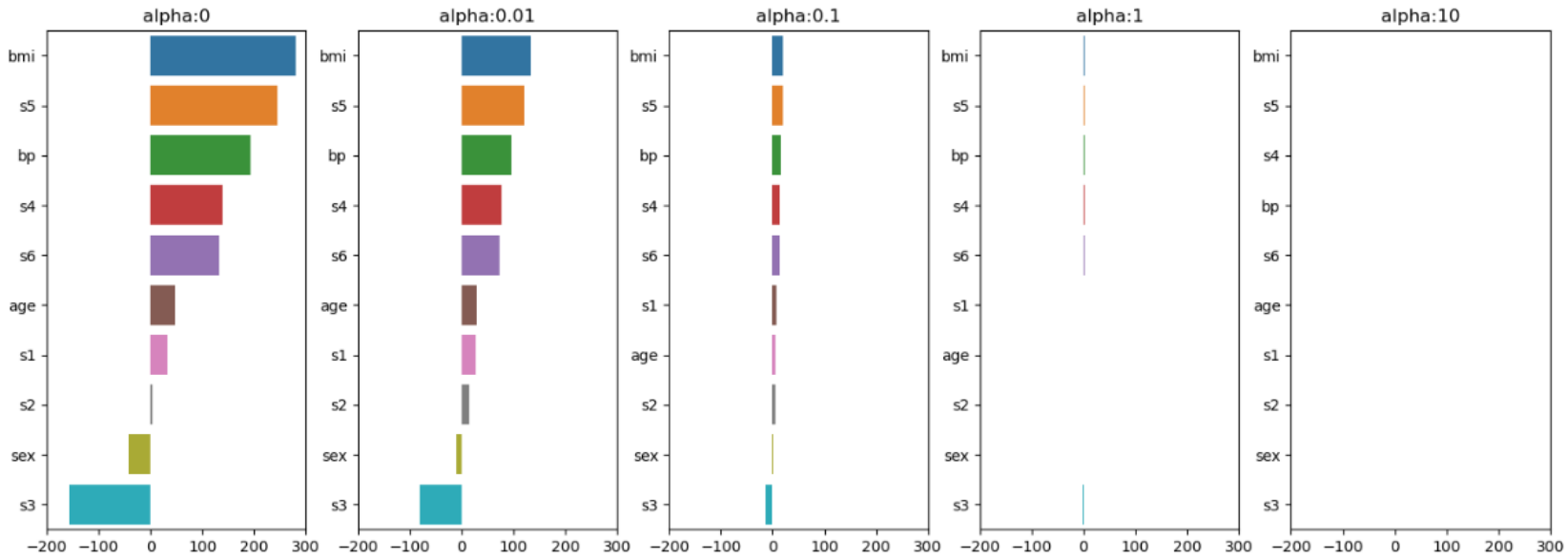
> 확률적 경사 하강법(SGD) 실습

```
fig, axs = plt.subplots(figsize = (18,6), nrows=1, ncols=5)
coeff_df = pd.DataFrame()
for pos, alpha in enumerate(alphas):
    sgd_reg = SGDRegressor(alpha = alpha)
    sgd_reg.fit(X, y)
    coeff = pd.Series(data = sgd_reg.coef_, index = X.columns)
    colname='alpha:'+ str(alpha)
    coeff_df[colname] = coeff
    coeff = coeff.sort_values(ascending = False)
    axs[pos].set_title(colname); axs[pos].set_xlim(-200, 300)
    sns.barplot(x = coeff.values, y=coeff.index, ax=axs[pos])
plt.show()
```

• 경사 하강법

> 확률적 경사 하강법(SGD) 실습

Out :



• 경사 하강법

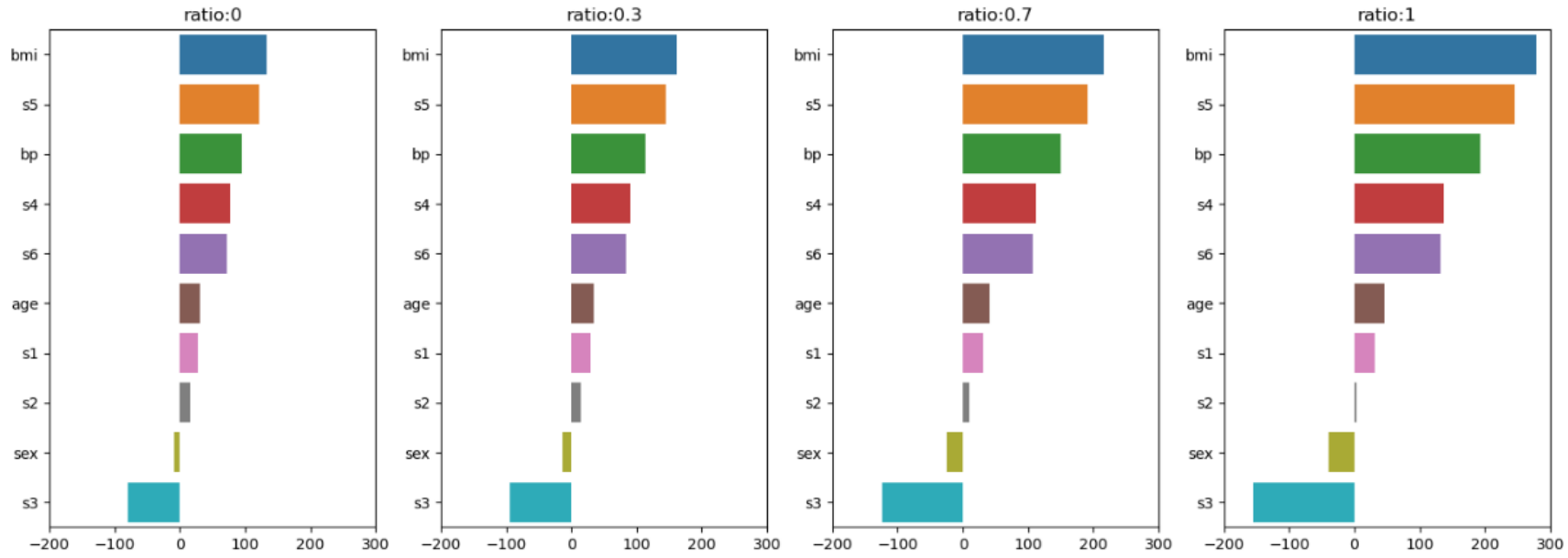
> 확률적 경사 하강법(SGD) 실습

```
fig, axs = plt.subplots(figsize = (18,6), nrows=1, ncols=4)
coeff_df = pd.DataFrame()
ratios = [0, 0.3, 0.7, 1]
for pos, ratio in enumerate(ratios):
    sgd_reg = SGDRegressor(alpha = 0.01, penalty='elasticnet', l1_ratio=ratio)
    sgd_reg.fit(X, y)
    coeff = pd.Series(data = sgd_reg.coef_, index = X.columns)
    colname='ratio:'+ str(ratio)
    coeff_df[colname] = coeff
    coeff = coeff.sort_values(ascending = False)
    axs[pos].set_title(colname); axs[pos].set_xlim(-200, 300)
    sns.barplot(x = coeff.values, y=coeff.index, ax=axs[pos])
plt.show()
```

• 경사 하강법

> 확률적 경사 하강법(SGD) 실습

Out :



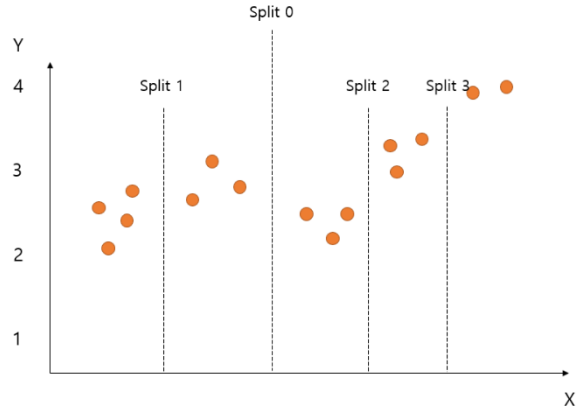
• 회귀 트리

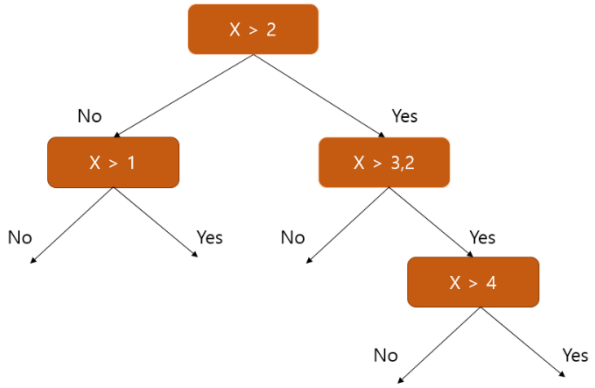
> 회귀 트리란 회귀 함수(선형, 비선형)를 기반으로 하지 않고 트리 기반으로 회귀 예측을 하는 방식

> 분류의 회귀와 크게 다르지 않지만 노드를 생성하는 방식이 다름

- 클래스 대신 노드에 속한 데이터의 평균으로 값을 다룸

> 분류에서 쓰이는 대부분의 트리 알고리즘들은 Classifier, Regression 2개로 구분되어 있다.





• 회귀 트리

> 회귀 트리 실습

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()

diabetes_df = pd.DataFrame(diabetes.data , columns = diabetes.feature_names)
diabetes_df['target'] = diabetes.target

y = diabetes_df.iloc[:, -1]
X = diabetes_df.iloc[:, :-1]
```

• 회귀 트리

> 회귀 트리 실습

```
rf_reg = RandomForestRegressor(random_state=0, n_estimators=1000)
mse_scores = cross_val_score(rf_reg, X, y, scoring="neg_mean_squared_error", cv = 5)
rmse_scores = np.sqrt(-1 * mse_scores)
avg_rmse = np.mean(rmse_scores)
print(' 5 교차 검증의 개별 Negative MSE scores: ',  np.round(mse_scores, 2))
print(' 5 교차 검증의 개별 RMSE scores : ', np.round(rmse_scores, 2))
print(f' 5 교차 검증의 평균 RMSE : {avg_rmse:.3f} ')
```

Out :

```
5 교차 검증의 개별 Negative MSE scores:  [-2994.11 -3112.26 -3547.44 -3272.69 -3709.92]
5 교차 검증의 개별 RMSE scores :  [54.72 55.79 59.56 57.21 60.91]
5 교차 검증의 평균 RMSE : 57.637
```

• 회귀 트리

> 회귀 트리 실습

```
def get_model_cv_prediction(model, X, y):
    neg_mse_scores = cross_val_score(model, X, y,
                                     scoring="neg_mean_squared_error", cv = 5)
    rmse_scores  = np.sqrt(-1 * neg_mse_scores)
    avg_rmse = np.mean(rmse_scores)
    print('##### ',model.__class__.__name__ , ' #####')
    print(' 5 교차 검증의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

• 회귀 트리

> 회귀 트리 실습

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor


dt_reg = DecisionTreeRegressor(random_state=0, max_depth=4)
gb_reg = GradientBoostingRegressor(random_state=0, n_estimators=1000)
xgb_reg = XGBRegressor(random_state=0, n_estimators=1000)
```

• 회귀 트리

> 회귀 트리 실습

```
models = [dt_reg, rf_reg, gb_reg, xgb_reg]
for model in models:
    get_model_cv_prediction(model, X, y)
```

Out :

```
#####  DecisionTreeRegressor  #####
 5 교차 검증의 평균 RMSE : 64.320
#####  RandomForestRegressor  #####
 5 교차 검증의 평균 RMSE : 57.637
#####  GradientBoostingRegressor  #####
 5 교차 검증의 평균 RMSE : 65.495
#####  XGBRegressor  #####
 5 교차 검증의 평균 RMSE : 63.035
```

• 회귀 트리

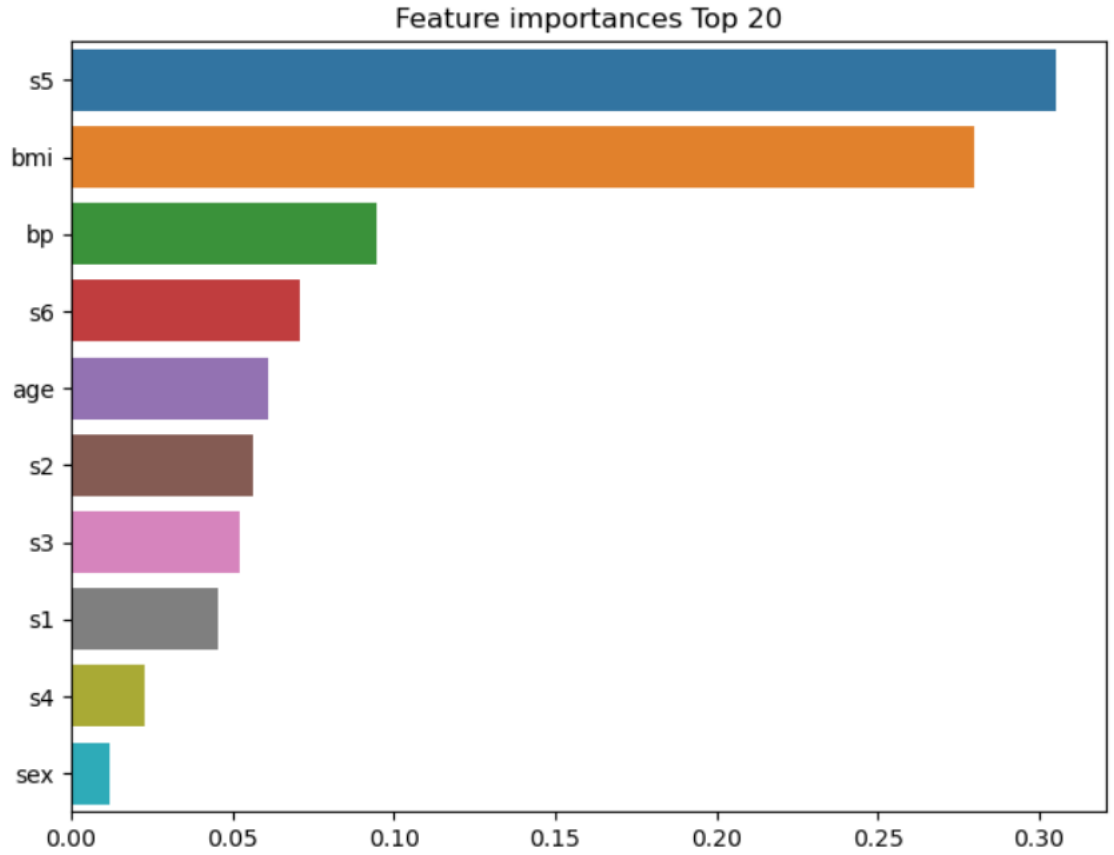
> 회귀 트리 동작 비교

```
rf_reg.fit(X, y)
ftr_importances_values = rf_reg.feature_importances_
ftr_importances = pd.Series(ftr_importances_values,index=X.columns)
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
plt.figure(figsize=(8,6))
plt.title('Feature importances Top 20')
sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
```

• 회귀 트리

> 회귀 트리 동작 비교

Out :



• 회귀 트리

> 회귀 트리 동작 비교

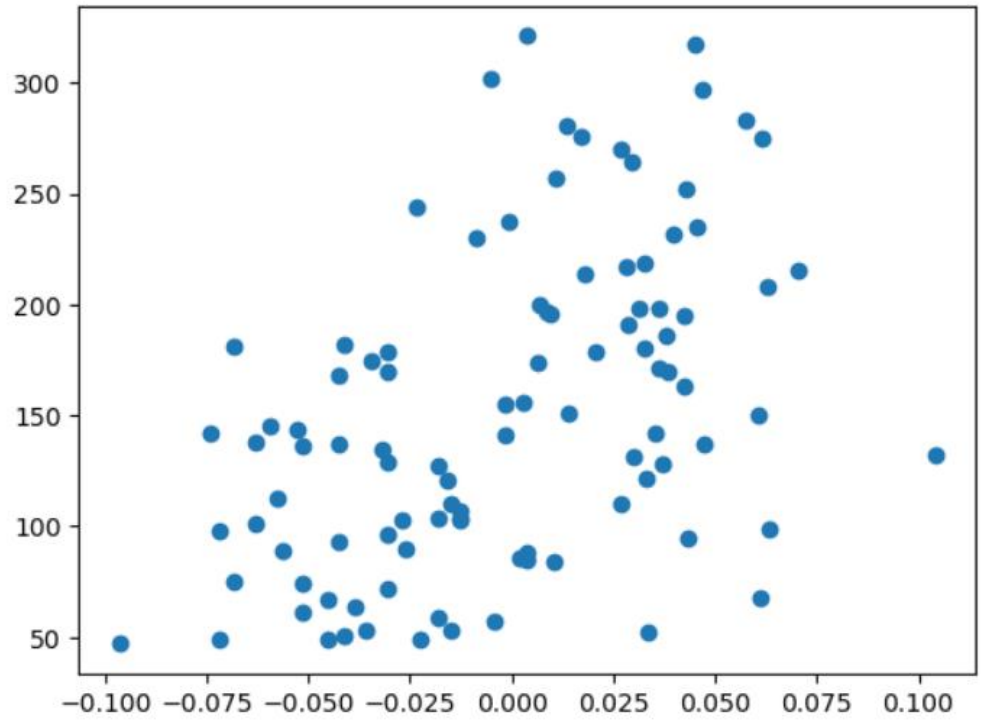
```
import matplotlib.pyplot as plt


sample_df = diabetes_df[['s5','target']]
sample_df = sample_df.sample(n=100,random_state=0)
print(sample_df.shape)
plt.figure()
plt.scatter(sample_df['s5'] , sample_df['target'])
```

Out : (100, 2)

• 회귀 트리

> 회귀 트리 동작 비교

Out :



• 회귀 트리

> 회귀 트리 동작 비교

```
import numpy as np
from sklearn.linear_model import LinearRegression

lr_reg = LinearRegression()
dt_reg = DecisionTreeRegressor()

X_test = np.linspace(-0.1, 0.1, 50).reshape(-1,1)
X_feature = sample_df['s5'].values.reshape(-1,1)
y_target = sample_df['target'].values.reshape(-1,1)

lr_reg.fit(X_feature, y_target)
dt_reg.fit(X_feature, y_target)
```

• 회귀 트리

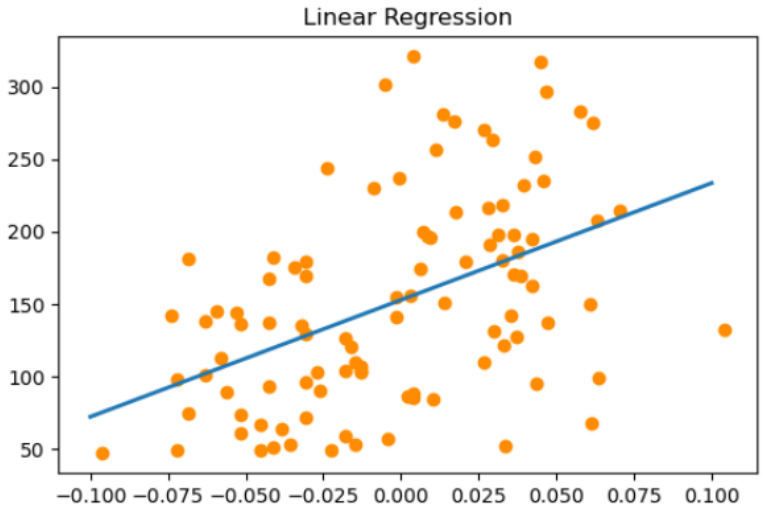
> 회귀 트리 동작 비교

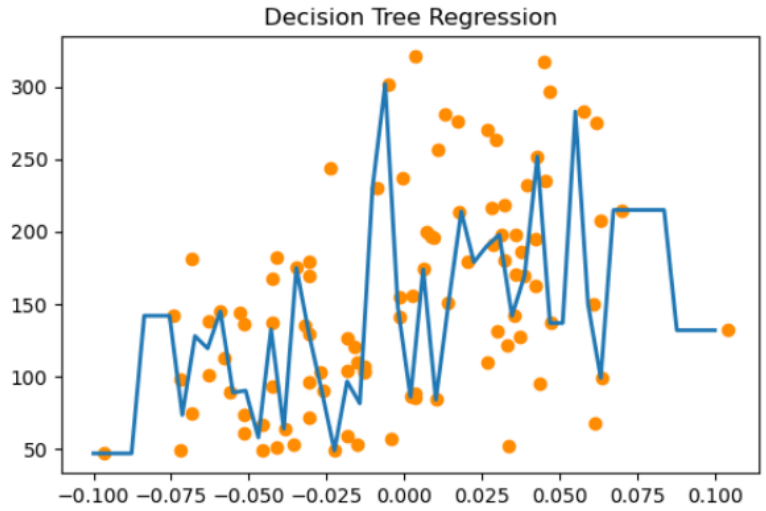
```
pred_lr = lr_reg.predict(X_test)
pred_dt = dt_reg.predict(X_test)

fig , (ax1, ax2) = plt.subplots(figsize=(14,4), ncols=2)
ax1.set_title('Linear Regression')
ax1.scatter(sample_df['s5'], sample_df['target'],  c="darkorange")
ax1.plot(X_test, pred_lr, linewidth=2 )
ax2.set_title('Decision Tree Regression')
ax2.scatter(sample_df['s5'], sample_df['target'],  c="darkorange")
ax2.plot(X_test, pred_dt, linewidth=2 )
```

• 회귀 트리

> 회귀 트리 동작 비교

Out :





• 회귀 트리

> 회귀 트리 동작 비교

```
dt_reg1 = DecisionTreeRegressor(max_depth=3)
dt_reg1.fit(X_feature, y_target)
pred_dt1 = dt_reg1.predict(X_test)
dt_reg2 = DecisionTreeRegressor(max_depth=5)
dt_reg2.fit(X_feature, y_target)
pred_dt2 = dt_reg2.predict(X_test)
```

• 회귀 트리

> 회귀 트리 동작 비교

```
fig , (ax1, ax2) = plt.subplots(figsize=(14,4), ncols=2)
ax1.set_title('Decision Tree Regression₩nMax_depth = 3')
ax1.scatter(sample_df['s5'], sample_df['target'],  c="darkorange")
ax1.plot(X_test, pred_dt1, linewidth=2 )
ax2.set_title('Decision Tree Regression₩nMax_depth = 5')
ax2.scatter(sample_df['s5'], sample_df['target'],  c="darkorange")
ax2.plot(X_test, pred_dt2, linewidth=2 )
```

• 회귀 트리

> 회귀 트리 동작 비교

Out :