5장 회귀

01 회귀 소개

회귀는 여러개의 독립변수와 한 개의 종속변수 간의 상관관계를 모델링하는 기법
독립변수는 피처에 해당하며 종속변수는 결정 값
머신 러닝 회귀 예측 핵심은 주어진 피처와 결정 값 데이터 기반에서 학습을 통해 최적의 회귀 계수를 찾아내는 것

지도학습은 분류와 회귀로 나뉨
분류의 예측값은 이산형 클래스 값이고 회귀는 연속형 숫자 값이라는 것이 가장 큰 차이

선형회귀는 실제 값과 예측값의 차이를 최소화하는 직선형 회귀선을 최적화하는 방식

<대표적인 선형 회귀 모델>

- 일반 선형 회귀: 예측값과 실제 값의 RSS를 최소화 할 수 있도록 회귀 계수를 최적화하며, 규제를 적용하지 않은 모델
- 릿지: 선형 회귀에 L2 규제를 추가한 회귀 모델(L2는 상대적으로 큰 회귀 계수 값의 예측 영향도를 감소시키기 위해 회귀 계수값을 더 작게 만드는 규제 모델)
- 라쏘: 선형 회귀에 L1 규제를 적용한 방식(L1은 예측 영향력이 작은 피처의 회귀 계수를 0으로 만들어 회귀 예측 시 피처가 선택되지 않게 하는 것)
- 엘라스틱넷: L1,L2 규제를 함께 결합한 모델로 주로 피처가 많은 데이터 세트에서 적용되며 L1 규제로 피처의 개수를 줄임과 동시에 L2 규제로 계수 값의 크기를 조정
- 로지스틱 회귀: 분류에 사용되는 선형 모델

02 단순 선형 회귀를 통한 회귀 이해

단순 선형 회귀는 독립변수도 하나, 종속변수도 하나인 선형 회귀
실제 값과 회귀 모델의 차이에 따른 오류 값을 남은 오류, 즉 잔차라고 부름
최적의 회귀모델을 만든다는 것은 바로 전체 데이터의 잔차 합이 최소가 되는 모델을 만든다는 의미
보통 오류 합을 계산 할 때는 절댓값을 취해서 더하거나(Mean Absolute Error) 오류 값의 제곱을 구해서 더하는 방식(RSS, Residual Sum of Square)을 취하고 일반적으로 RSS 방식을 씀

회귀에서 RSS는 비용이며 w변수로 구성되는 RSS를 비용함수 라고 함
머신 러닝 회귀 알고리즘은 데이터를 계속 학습하면서 이 비용함수가 반환하는 값을 지속해서 감소시키고 최종적으로는 더 이상 감소하지 않는 최소의 오류 값을 구하는 것
비용 함수를 손실 함수라고도 함

03 비용 최소화하기 - 경사 하강법

경사하강법은 반복적으로 비용 함수의 반환 값, 즉 예측값과 실제 값의 차이가 작아지는 방향성을 가지고 w파라미터를 지속해서 보정해 나감
비용함수가 포물선 형태의 2차 함수라면 경사 하강법은 최초 w에서부터 미분을 적용한 뒤 이 미분 값이 계속 감소하는 방향으로 순차적으로 w를 업데이트함
더 이상 미분된 1차 함수의 기울기가 감소하지 않는 지점을 비용 함수가 최소인 지점으로 간주하고 그때의 w를 반환

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(0)
X = 2 * np.random.rand(100,1)
y = 6 +4 * X+ np.random.randn(100,1)

plt.scatter(X, y)
```

```
#비용함수
def get_cost(y, y_pred):
    N=len(y)
    cost=np.sum(np.square(y-y_pred))/N
    return cost
```

```
#경사하강법
def get_weight_updates(w1, w0, X, y, learning_rate=0.01):
    N = len(y)
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)
    y_pred = np.dot(X, w1.T) + w0
    diff = y-y_pred

    w0_factors = np.ones((N,1))

    w1_update = -(2/N)*learning_rate*(np.dot(X.T, diff))
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T, diff))

    return w1_update, w0_update
```

```
def gradient_descent_steps(X, y, iters=10000):
    w0 = np.zeros((1,1))
    w1 = np.zeros((1,1))

    for ind in range(iters):
        w1_update, w0_update = get_weight_updates(w1, w0, X, y, learning_rate=0.01)
        w1 = w1 - w1_update
        w0 = w0 - w0_update

    return w1, w0
```

```
w1, w0 = gradient_descent_steps(X, y, iters=1000)
print("w1:{0:.3f} w0:{1:.3f}".format(w1[0,0], w0[0,0]))
y_pred = w1[0,0] * X + w0
print('Gradient Descent Total Cost:{0:.4f}'.format(get_cost(y, y_pred)))
```

```
w1:4.022 w0:6.162
Gradient Descent Total Cost:0.9935
```

```
plt.scatter(X, y)
plt.plot(X,y_pred)
```

실전에서는 대부분 확률적 경사 하강법 이용
확률적 경사 하강법: 전체 입력 데이터로 w가 업데이트되는 값을 계산하는 것이 아니라 일부 데이터만 이용해 w가 업데이트 되는 값을 계산하므로 빠른 속도 보장
대용량 데이터경우 대부분 확률적 경사 하강법이나 미니 배치 확률적 경사 하강법 이용해 최적 비용함수 도출

```
#확률적 경사 하강법
def stochastic_gradient_descent_steps(X, y, batch_size=10, iters=1000):
    w0 = np.zeros((1,1))
    w1 = np.zeros((1,1))
    prev_cost = 100000
    iter_index =0

    for ind in range(iters):
        np.random.seed(ind)
        stochastic_random_index = np.random.permutation(X.shape[0])
        sample_X = X[stochastic_random_index[0:batch_size]]
        sample_y = y[stochastic_random_index[0:batch_size]]
        w1_update, w0_update = get_weight_updates(w1, w0, sample_X, sample_y, learning_rate=0.(
        w1 = w1 - w1_update
        w0 = w0 - w0_update

    return w1, w0
```

```
w1, w0 = stochastic_gradient_descent_steps(X, y, iters=1000)
print("w1:",round(w1[0,0],3),"w0:",round(w0[0,0],3))
y_pred = w1[0,0] * X + w0
print('Stochastic Gradient Descent Total Cost:{0:.4f}'.format(get_cost(y, y_pred)))

```

```
w1: 4.028 w0: 6.156
Stochastic Gradient Descent Total Cost:0.9937
```

04 사이킷런 LinearRegression을 이용한 보스턴 주택 가격 예측

LinearRegression 클래스 - Ordinary Least Squares

class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False,copy_X=True,n_jobs=1)

- fit_intercept: 절편을 계산할 것인지 말지 지정(디폴트 true)
- normalize: 회귀 수행전 입력 데이터 세트 정규화(디폴트 false)

- coef_: fit() 메서드 수행했을 때 회귀 계수가 배열 형태로 저장하는 속성
- intercept: intercept 값

ordinary least squares기반의 회귀 계수 계산은 입력 피처의 독립성에 많은 영향 받음
피처 간 상관관계가 매우 높은 경우 분산이 매우 커져 오류에 민감해짐
이러한 현상을 다중공선성 문제라고함
일반적으로 상관관계가 높은 피처가 많은 경우 독립적인 중요한 피처만 남기고 제거하거나 규제를 적용하거나 PCA통해 차원 축소 수행

<회귀 평가 지표>

- MAE:mean absolute erroer, 실제 값과 예측 값 차이를 절댓값으로 변환해 평균
- MSE:mean squared error, 실제 값과 예측값의 차이를 제곱해 평균
- RMSE: MSE에 루트 씌운 것
- R2: 분산 기반, 실제 값의 분산 대비 예측 값의 분산 비율,1에 가까울 수록 정확도 높음

LinearRegression을 이용해 보스턴 주택 가격 회귀 구현

```
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_boston
%matplotlib inline

boston = load_boston()

bostonDF = pd.DataFrame(boston.data , columns = boston.feature_names)

bostonDF['PRICE'] = boston.target
print('Boston 데이타셋 크기 :',bostonDF.shape)
bostonDF.head()
```

regplot(): x,y축 값의 산점도와 함께 선형 회귀 직선을 그려줌
matplotlib.subplots():각 ax마다 칼럼과 price 관계 표현

```
fig, axs = plt.subplots(figsize=(16,8) , ncols=4 , nrows=2)
lm_features = ['RM','ZN','INDUS','NOX','AGE','PTRATIO','LSTAT','RAD']
for i , feature in enumerate(lm_features):
    row = int(i/4)
    col = i%4
    sns.regplot(x=feature , y='PRICE',data=bostonDF , ax=axs[row][col])
```

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

y_target = bostonDF['PRICE']
X_data = bostonDF.drop(['PRICE'],axis=1,inplace=False)

X_train , X_test , y_train , y_test = train_test_split(X_data , y_target ,test_size=0.3, rando

# Linear Regression OLS로 학습/예측/평가 수행.
lr = LinearRegression()
lr.fit(X_train ,y_train )
y_preds = lr.predict(X_test)
mse = mean_squared_error(y_test, y_preds)
rmse = np.sqrt(mse)

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_preds)))
```

```
MSE : 17.297 , RMSE : 4.159
Variance score : 0.757
```

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

```
절편 값: 40.995595172164755
회귀 계수값: [ -0.1   0.1   0.    3.  -19.8   3.4   0.   -1.7   0.4  -0.   -0.9   0.
  -0.6]
```

```
coeff = pd.Series(data=np.round(lr.coef_, 1), index=X_data.columns )
coeff.sort_values(ascending=False)
```

```
RM          3.4
CHAS        3.0
RAD         0.4
ZN          0.1
INDUS       0.0
AGE         0.0
```

```
TAX       -0.0
B          0.0
CRIM      -0.1
LSTAT     -0.6
PTRATIO   -0.9
DIS       -1.7
NOX      -19.8
dtype: float64
```

```
from sklearn.model_selection import cross_val_score

y_target = bostonDF['PRICE']
X_data = bostonDF.drop(['PRICE'],axis=1,inplace=False)
lr = LinearRegression()

neg_mse_scores = cross_val_score(lr, X_data, y_target, 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, 2))
print(' 5 folds 의 개별 RMSE scores : ', np.round(rmse_scores, 2))
print(' 5 folds 의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

```
 5 folds 의 개별 Negative MSE scores:  [-12.46 -26.05 -33.07 -80.76 -33.31]
 5 folds 의 개별 RMSE scores :  [3.53 5.1  5.75 8.99 5.77]
 5 folds 의 평균 RMSE : 5.829
```

05 다항 회귀와 과적합/과소적합 이해

다항 회귀 이해

회귀가 독립변수의 단항식이 아닌 2차, 3차 방정식과 같은 다항식으로 표현되는 것을 다항 회귀라고 함
다항 회귀는 선형 회귀임

```
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)
```

```
일차 단항식 계수 feature:
 [[0 1]
 [2 3]]
```

```
변환된 2차 다항식 계수 feature:
 [[1. 0. 1. 0. 0. 1.]
 [1. 2. 3. 4. 6. 9.]]
```

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

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

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

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

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

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
import numpy as np

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

model = Pipeline([('poly', PolynomialFeatures(degree=3)),
                  ('linear', LinearRegression())])
X = np.arange(4).reshape(2,2)
y = polynomial_func(X)

model = model.fit(X, y)
```

```
print('Polynomial 회귀 계수Wn', np.round(model.named_steps['linear'].coef_, 2))

```

```
Polynomial 회귀 계수
 [0.   0.18 0.18 0.36 0.54 0.72 0.72 1.08 1.62 2.34]
```

다항회귀를 이용한 과소적합 및 과적합 이해

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

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

```

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

for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("linear_regression", linear_regression)])
    pipeline.fit(X.reshape(-1, 1), y)

    scores = cross_val_score(pipeline, X.reshape(-1,1), y,scoring="neg_mean_squared_error", cv=
    coefficients = pipeline.named_steps['linear_regression'].coef_
    print('WnDegree {0} 회귀 계수는 {1} 입니다.'.format(degrees[i], np.round(coefficients),2))
    print('Degree {0} MSE 는 {1:.2f} 입니다.'.format(degrees[i] , -1*np.mean(scores)))

    X_test = np.linspace(0, 1, 100)

    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")

    plt.plot(X_test, true_fun(X_test), '--', label="True function")
    plt.scatter(X, y, edgecolor='b', s=20, label="Samples")

    plt.xlabel("x"); plt.ylabel("y"); plt.xlim((0, 1)); plt.ylim((-2, 2)); plt.legend(loc="bes
```

```
    plt.title("Degree {}WnMSE = {:.2e}(+/- {:.2e})".format(degrees[i], -scores.mean(), scores.s

plt.show()
```

첫번째는 과소적합모델, 고편향성 가짐
세번째는 과적합이 심한 모델, 고분산성 가짐

편향 분산 트레이드오프

일반적으로 편향과 분산은 한 쪽이 높으면 한 쪽이 낮아지는 경향
높은 편향/낮은 분산에서 과소적합되기 쉬우며 낮은 편향/높은 분산에서 과적합되기 쉬움

06 규제 선형 모델 - 릿지, 라쏘, 엘라스틱넷

규제 선형 모델의 개요

비용함수는 RSS 최소화 방법과 과적합을 방지하기 위해 회귀 계수 값이 커지지 않도록 하는 방법이 서로 균형을 이뤄야함

비용함수 목표 =Min(RSS(W)+alpha*||W||2)
alpha는 학습 데이터 적합 정도와 회귀 계수 값의 크기 제어를 수행하는 튜닝 파라미터
alpha=0인 경우 W가 커도 비용함수는 Min(RSS(W))
alpha=무한대인경우 비용함수는 W를 0에 가깝게 최소화 해야함

alpha를 0에서 부터 지속적으로 값을 증가시키면 회귀 계수 값의 크기를 감소시킬 수 있음
규제: 비용함수에 alpha값으로 페널티를 부여해 회귀 계수 값의 크기를 감소시켜 과적합을 개선하는 방식
L2규제: W의 제곱에 페널티를 부여하는 방식 -> 릿지 회귀
L1규제: W의 절댓값에 대해 페널티 부여 -> 라쏘 회귀

릿지 회귀

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

ridge = Ridge(alpha = 10)
neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring="neg_mean_squared_error", cv
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))
```

```
 5 folds 의 개별 Negative MSE scores:  [-11.422 -24.294 -28.144 -74.599 -28.517]
 5 folds 의 개별 RMSE scores :  [3.38  4.929 5.305 8.637 5.34 ]
 5 folds 의 평균 RMSE : 5.518
```

```
alphas = [0 , 0.1 , 1 , 10 , 100]

for alpha in alphas :
    ridge = Ridge(alpha = alpha)

    neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring="neg_mean_squared_error"
    avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
    print('alpha {0} 일 때 5 folds 의 평균 RMSE : {1:.3f} '.format(alpha,avg_rmse))
```

```
alpha 0 일 때 5 folds 의 평균 RMSE : 5.829
alpha 0.1 일 때 5 folds 의 평균 RMSE : 5.788
alpha 1 일 때 5 folds 의 평균 RMSE : 5.653
alpha 10 일 때 5 folds 의 평균 RMSE : 5.518
alpha 100 일 때 5 folds 의 평균 RMSE : 5.330
```

```
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_data , y_target)

    coeff = pd.Series(data=ridge.coef_ , index=X_data.columns )
    colname='alpha:'+str(alpha)
    coeff_df[colname] = coeff

    coeff = coeff.sort_values(ascending=False)
    axs[pos].set_title(colname)
    axs[pos].set_xlim(-3,6)
    sns.barplot(x=coeff.values , y=coeff.index, ax=axs[pos])

plt.show()

```

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

라쏘 회귀

```
from sklearn.linear_model import Lasso, ElasticNet

def get_linear_reg_eval(model_name, params=None, X_data_n=None, y_target_n=None,
                        verbose=True, return_coeff=True):
    coeff_df = pd.DataFrame()
    if verbose : print('####### ', model_name , '#######')
    for param in params:
        if model_name =='Ridge': model = Ridge(alpha=param)
        elif model_name =='Lasso': model = Lasso(alpha=param)
        elif model_name =='ElasticNet': model = ElasticNet(alpha=param, l1_ratio=0.7)
        neg_mse_scores = cross_val_score(model, X_data_n,
                                         y_target_n, scoring="neg_mean_squared_error", cv =
        avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
        print('alpha {0}일 때 5 폴드 세트의 평균 RMSE: {1:.3f} '.format(param, avg_rmse))
        # cross_val_score는 evaluation metric만 반환하므로 모델을 다시 학습하여 회귀 계수 추출

        model.fit(X_data_n , y_target_n)
        if return_coeff:
            # alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가
            coeff = pd.Series(data=model.coef_ , index=X_data_n.columns )
            colname='alpha:'+str(param)
            coeff_df[colname] = coeff

    return coeff_df

```

```
lasso_alphas = [ 0.07, 0.1, 0.5, 1, 3]
coeff_lasso_df =get_linear_reg_eval('Lasso', params=lasso_alphas, X_data_n=X_data, y_target_n=y
```

```
#######  Lasso #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 5.612
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.615
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 5.669
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.776
alpha 3일 때 5 폴드 세트의 평균 RMSE: 6.189
```

```
1 sort_column = 'alpha:'+str(lasso_alphas[0])
2 coeff_lasso_df.sort_values(by=sort_column, ascending=False)
```

엘라스틱넷 회귀

```
1 elastic_alphas = [ 0.07, 0.1, 0.5, 1, 3]
2 coeff_elastic_df =get_linear_reg_eval('ElasticNet', params=elastic_alphas,
3                                       X_data_n=X_data, y_target_n=y_target)
```

```
#######  ElasticNet #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 5.542
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.526
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 5.467
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.597
alpha 3일 때 5 폴드 세트의 평균 RMSE: 6.068
```

```
1 sort_column = 'alpha:'+str(elastic_alphas[0])
2 coeff_elastic_df.sort_values(by=sort_column, ascending=False)
```

선형 회귀 모델을 위한 데이터 변환

1.standardscaler 클래스를 이용해 평균이 0, 분산이 1인 표준 정규 분포를 가진 데이터 세트로 변환하거나 minmaxscaler클래스를 이용해 최솟값이 0이고 최댓값이 1인 값으로 정규화 수행
2.스케일링/정규화 수행한 데이터 세트에 다시 다항 특성을 적용하여 변환하는 방법
3.로그변환

타깃 값의 경우는 일반적으로 로그 변환 적용

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures

def get_scaled_data(method='None', p_degree=None, input_data=None):
    if method == 'Standard':
        scaled_data = StandardScaler().fit_transform(input_data)
    elif method == 'MinMax':
        scaled_data = MinMaxScaler().fit_transform(input_data)
    elif method == 'Log':
        scaled_data = np.log1p(input_data)
    else:
        scaled_data = input_data

    if p_degree != None:
        scaled_data = PolynomialFeatures(degree=p_degree,
                                         include_bias=False).fit_transform(scaled_data)

    return scaled_data
```

```
alphas = [0.1, 1, 10, 100]

scale_methods=[(None, None), ('Standard', None), ('Standard', 2),
               ('MinMax', None), ('MinMax', 2), ('Log', None)]
for scale_method in scale_methods:
    X_data_scaled = get_scaled_data(method=scale_method[0], p_degree=scale_method[1],
                                    input_data=X_data)
```

```
    print(X_data_scaled.shape, X_data.shape)
    print('Wn## 변환 유형:{0}, Polynomial Degree:{1}'.format(scale_method[0], scale_method[1])
    get_linear_reg_eval('Ridge', params=alphas, X_data_n=X_data_scaled,
                        y_target_n=y_target, verbose=False, return_coeff=False)
```

```
(506, 13) (506, 13)

## 변환 유형:None, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.788
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.653
alpha 10일 때 5 폴드 세트의 평균 RMSE: 5.518
alpha 100일 때 5 폴드 세트의 평균 RMSE: 5.330
(506, 13) (506, 13)

## 변환 유형:Standard, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.826
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.803
alpha 10일 때 5 폴드 세트의 평균 RMSE: 5.637
alpha 100일 때 5 폴드 세트의 평균 RMSE: 5.421
(506, 104) (506, 13)

## 변환 유형:Standard, Polynomial Degree:2
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 8.827
alpha 1일 때 5 폴드 세트의 평균 RMSE: 6.871
alpha 10일 때 5 폴드 세트의 평균 RMSE: 5.485
alpha 100일 때 5 폴드 세트의 평균 RMSE: 4.634
(506, 13) (506, 13)

## 변환 유형:MinMax, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.764
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.465
alpha 10일 때 5 폴드 세트의 평균 RMSE: 5.754
alpha 100일 때 5 폴드 세트의 평균 RMSE: 7.635
(506, 104) (506, 13)

## 변환 유형:MinMax, Polynomial Degree:2
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.298
alpha 1일 때 5 폴드 세트의 평균 RMSE: 4.323
alpha 10일 때 5 폴드 세트의 평균 RMSE: 5.185
alpha 100일 때 5 폴드 세트의 평균 RMSE: 6.538
(506, 13) (506, 13)

## 변환 유형:Log, Polynomial Degree:None
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 4.770
alpha 1일 때 5 폴드 세트의 평균 RMSE: 4.676
alpha 10일 때 5 폴드 세트의 평균 RMSE: 4.836
alpha 100일 때 5 폴드 세트의 평균 RMSE: 6.241
```

07 로지스틱 회귀

시그모이드 함수 y=1/(1+e^(-x))

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
4
5 from sklearn.datasets import load_breast_cancer
6 from sklearn.linear_model import LogisticRegression
7
8 cancer = load_breast_cancer()
```

```
1 from sklearn.preprocessing import StandardScaler
2 from sklearn.model_selection import train_test_split
3
4 scaler = StandardScaler()
5 data_scaled = scaler.fit_transform(cancer.data)
6
7 X_train , X_test, y_train , y_test = train_test_split(data_scaled, cancer.target, test_size=0.3
```

```
1 from sklearn.metrics import accuracy_score, roc_auc_score
2
3 lr_clf = LogisticRegression()
4 lr_clf.fit(X_train, y_train)
5 lr_preds = lr_clf.predict(X_test)
6
7 print('accuracy: {:0.3f}'.format(accuracy_score(y_test, lr_preds)))
8 print('roc_auc: {:0.3f}'.format(roc_auc_score(y_test , lr_preds)))
```

```
accuracy: 0.977
roc_auc: 0.972
```

```
1 from sklearn.model_selection import GridSearchCV
2
3 params={'penalty':['l2', 'l1'],
4         'C':[0.01, 0.1, 1, 1, 5, 10]}
5
6 grid_clf = GridSearchCV(lr_clf, param_grid=params, scoring='accuracy', cv=3 )
7 grid_clf.fit(data_scaled, cancer.target)
8 print('최적 하이퍼 파라미터:{0}, 최적 평균 정확도:{1:.3f}'.format(grid_clf.best_params_,
9                                                    grid_clf.best_score_))
```

```
최적 하이퍼 파라미터:{'C': 1, 'penalty': 'l2'}, 최적 평균 정확도:0.975
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:372: FitFailedW
18 fits failed out of a total of 36.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
18 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", line 1461,
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", line 449,
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:972: UserWarning: C
 0.97539218        nan 0.97011974        nan 0.96661097        nan]
  category=UserWarning,
```

08 회귀 트리

회귀를 위한 트리를 생성하고 이를 기반으로 회귀 예측을 하는 것

분류 트리가 특정 클래스 레이블을 결정하는 것과는 달리 회귀 트리는 리프 노드에 속한 데이터 값의 평균값을 구해 회귀 예측 값을 계산

```
 1 from sklearn.datasets import load_boston
 2 from sklearn.model_selection import cross_val_score
 3 from sklearn.ensemble import RandomForestRegressor
 4 import pandas as pd
 5 import numpy as np
 6
 7 boston = load_boston()
 8 bostonDF = pd.DataFrame(boston.data, columns = boston.feature_names)
 9
10 bostonDF['PRICE'] = boston.target
11 y_target = bostonDF['PRICE']
12 X_data = bostonDF.drop(['PRICE'], axis=1,inplace=False)
13
14 rf = RandomForestRegressor(random_state=0, n_estimators=1000)
15 neg_mse_scores = cross_val_score(rf, X_data, y_target, scoring="neg_mean_squared_error", cv = 5
16 rmse_scores  = np.sqrt(-1 * neg_mse_scores)
17 avg_rmse = np.mean(rmse_scores)
18
19 print(' 5 교차 검증의 개별 Negative MSE scores: ', np.round(neg_mse_scores, 2))
20 print(' 5 교차 검증의 개별 RMSE scores : ', np.round(rmse_scores, 2))
21 print(' 5 교차 검증의 평균 RMSE : {0:.3f} '.format(avg_rmse))
22
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Functi

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np
```

```
            data_url = "http://lib.stat.cmu.edu/datasets/boston"
            raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
            data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
            target = raw_df.values[1::2, 2]

        Alternative datasets include the California housing dataset (i.e.
        :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
        dataset. You can load the datasets as follows::

            from sklearn.datasets import fetch_california_housing
            housing = fetch_california_housing()

        for the California housing dataset and::

            from sklearn.datasets import fetch_openml
            housing = fetch_openml(name="house_prices", as_frame=True)

        for the Ames housing dataset.

      warnings.warn(msg, category=FutureWarning)
    5 교차 검증의 개별 Negative MSE scores:  [ -7.88 -13.14 -20.57 -46.23 -18.88]
    5 교차 검증의 개별 RMSE scores :  [2.81 3.63 4.54 6.8  4.34]
    5 교차 검증의 평균 RMSE : 4.423
```

```
def get_model_cv_prediction(model, X_data, y_target):
    neg_mse_scores = cross_val_score(model, X_data, y_target, scoring="neg_mean_squared_error"
    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))
```

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

dt_reg = DecisionTreeRegressor(random_state=0, max_depth=4)
rf_reg = RandomForestRegressor(random_state=0, n_estimators=1000)
gb_reg = GradientBoostingRegressor(random_state=0, n_estimators=1000)
xgb_reg = XGBRegressor(n_estimators=1000)
lgb_reg = LGBMRegressor(n_estimators=1000)


models = [dt_reg, rf_reg, gb_reg, xgb_reg, lgb_reg]
for model in models:
    get_model_cv_prediction(model, X_data, y_target)
```

```
#####  DecisionTreeRegressor  #####
 5 교차 검증의 평균 RMSE : 5.978
#####  RandomForestRegressor  #####
 5 교차 검증의 평균 RMSE : 4.423
#####  GradientBoostingRegressor  #####
 5 교차 검증의 평균 RMSE : 4.269
[10:58:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecat
[10:58:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecat
```

```
[10:58:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:58:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:58:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
#####  XGBRegressor  #####
 5 교차 검증의 평균 RMSE : 4.089
#####  LGBMRegressor  #####
 5 교차 검증의 평균 RMSE : 4.646
```

```
import seaborn as sns
%matplotlib inline

rf_reg = RandomForestRegressor(n_estimators=1000)

rf_reg.fit(X_data, y_target)

feature_series = pd.Series(data=rf_reg.feature_importances_, index=X_data.columns )
feature_series = feature_series.sort_values(ascending=False)
sns.barplot(x= feature_series, y=feature_series.index)
```

```
import matplotlib.pyplot as plt
%matplotlib inline

bostonDF_sample = bostonDF[['RM','PRICE']]
bostonDF_sample = bostonDF_sample.sample(n=100,random_state=0)
print(bostonDF_sample.shape)
plt.figure()
plt.scatter(bostonDF_sample.RM , bostonDF_sample.PRICE,c="darkorange")
```

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

lr_reg = LinearRegression()
rf_reg2 = DecisionTreeRegressor(max_depth=2)
rf_reg7 = DecisionTreeRegressor(max_depth=7)

X_test = np.arange(4.5, 8.5, 0.04).reshape(-1, 1)

X_feature = bostonDF_sample['RM'].values.reshape(-1,1)
y_target = bostonDF_sample['PRICE'].values.reshape(-1,1)

lr_reg.fit(X_feature, y_target)
rf_reg2.fit(X_feature, y_target)
rf_reg7.fit(X_feature, y_target)

pred_lr = lr_reg.predict(X_test)
pred_rf2 = rf_reg2.predict(X_test)
pred_rf7 = rf_reg7.predict(X_test)
```

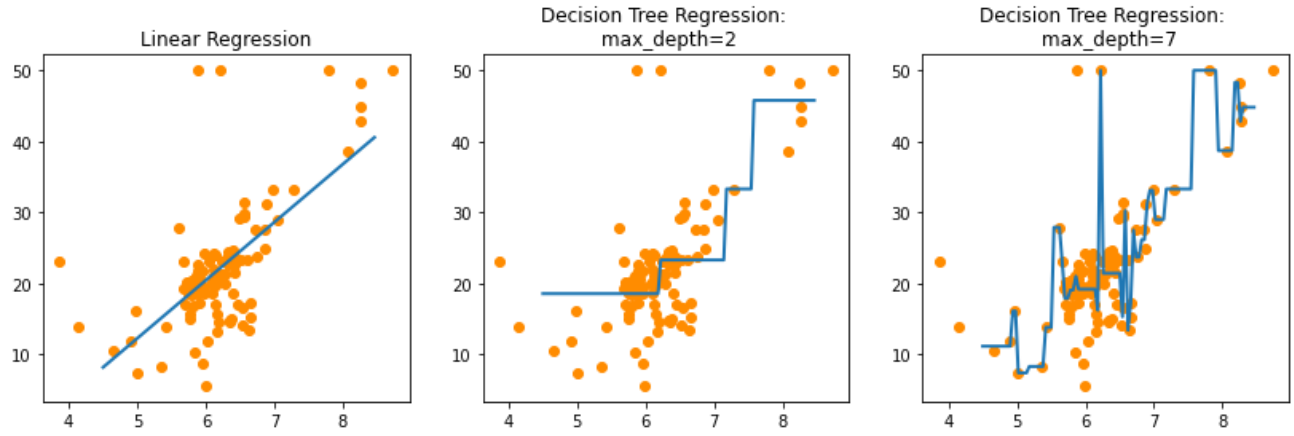
```
fig , (ax1, ax2, ax3) = plt.subplots(figsize=(14,4), ncols=3)

ax1.set_title('Linear Regression')
ax1.scatter(bostonDF_sample.RM, bostonDF_sample.PRICE, c="darkorange")
ax1.plot(X_test, pred_lr,label="linear", linewidth=2 )

ax2.set_title('Decision Tree Regression: ₩n max_depth=2')
ax2.scatter(bostonDF_sample.RM, bostonDF_sample.PRICE, c="darkorange")
ax2.plot(X_test, pred_rf2, label="max_depth:3", linewidth=2 )

ax3.set_title('Decision Tree Regression: ₩n max_depth=7')
ax3.scatter(bostonDF_sample.RM, bostonDF_sample.PRICE, c="darkorange")
ax3.plot(X_test, pred_rf7, label="max_depth:7", linewidth=2)
```

```
[<matplotlib.lines.Line2D at 0x7f7c95347550>]
```



✓ 1초 오후 8:00에 완료됨