

# CHAPTER 2

## Table of contents

### 1. 기본 통계량 계산

### 2. Sklearn for machine learning

### 3. Regression

- 1) Python for regression
- 2) Single linear regression
- 3) Multiple linear regression

### 4. Classification

- 1) Logistic regression
- 2) LDA

### 5. CART

- 1) Regression tree
- 2) Classification tree

### 6. Ensemble

- 1) Bagging
- 2) Boosting
- 3) Random forest

### 7. Study Case for classification

## 1. 기본 통계량 계산

통계량의 계산이나 기본적인 데이터 분석에 사용되는 함수는 numpy와 scipy 패키지에 들어있다. 우선, 필요한 패키지를 불러들인 후 fish\_data라는 변수에 10개의 데이터를 저장한다.

```
# import package
import numpy as np
import pandas as pd
import scipy as sp

# 예제 데이터 생성
fish_data = np.array([2, 3, 3, 4, 4, 4, 4, 5, 5, 6])
```

위의 예제 데이터를 이용하여 기본적인 통계량을 계산한다.

```
# sum
print(sum(fish_data))

# mean
print(np.mean(fish_data))

# variance
sigma = np.var(fish_data, ddof = 1) # ddof=0 이면 잔차의 합을 1/n 으로 나눈 경우
print(sigma)

# standard variance
print(np.sqrt(sigma))
print(np.std(fish_data, ddof = 1))

# Max, Min
print(np.max(fish_data))
print(np.min(fish_data))

# median
print(np.median(fish_data))

# quartile
print(np.quantile(fish_data, .25)) # first quartile
print(np.quantile(fish_data, .5)) # second quartile
print(np.quantile(fish_data, .75)) # third quartile

40
4.0
1.333333333333333
1.1547005383792515
1.1547005383792515
6
2
4.0
3.25
4.0
4.75
```

NOTE)

일반적으로 분산은 불편분산을 이용한다.

## 2. sklearn of machine learning

본격적으로 데이터 분석으로 들어가기 전에 python의 package인 sklearn의 built-in dataset에 대해 간단히 알아보도록 한다. sklearn에 내장되어 있는 dataset을 이용하기 위해서 다음을 참고한다.

### 1) Built – In dataset

Built – In dataset	
load_iris()	Load and return the iris dataset (classification)
load_diabetes()	Load and return the diabetes dataset (regression)
load_digits()	Load and return the digits dataset (classification)
load_linnerud()	Load and return the physical exercise Linnerud dataset
load_wine()	Load and return the wine dataset (classification)
load_breast_cancer()	Load and return the breast cancer wisconsin dataset (classification)

### 2) Dataset

built-in dataset은 sklearn.utils.Bunch라는 자료구조를 활용한다. key-value 형식으로 구성되어 있으며, dictionary 형 타입과 유사한 구조를 가진다. 공통 key는 다음과 같다.

NOTE)

R은 통계 분석에, python은 머신러닝과 딥러닝에 특화된 프로그램이므로 통계적 관점에서 몇 가지 용어들의 표현방식이 다르다는 것을 알아두자.

data	feature (설명변수, X), numpy 배열 형태 또는 Matrix
target	target (종속변수, Y), numpy 배열 형태 또는 Matrix
Feature_names	feature 데이터(x)의 이름
target_names	target 데이터(y)의 이름
DESCR	데이터 셋에 대한 설명
filename	데이터 셋의 파일 저장 위치(csv)

NOTE)

R의 경우 attributes(dataset)를 이용한다.

### 3. Regression

통계적 분석 방법은 크게 regression과 classification으로 나뉜다. regression은 Y(종속변수, target)가 numeric인 경우, classification은 Y가 categorical인 경우 이용한다. 방법은 다르지만 두 분석방법의 목적은 모두 예측값을 구하기 위함이다.

regression은 통계적 분석 방법의 하나로 데이터에 있는 독립변수들과 종속변수 사이의 관계를 모델링하여 예측값을 구하는 방법이다. regression은 크게 simple linear regression과 multiple linear regression으로 나뉜다. single linear regression의 경우 하나의 X(설명변수, feature)값을 이용하는 모델이고, multiple linear regression은 여러개의 X(multiple features)를 이용하여 Y를 예측하는 모델이다.

NOTE)

일반적으로 독립변수(설명변수)를 X, 종속변수(반응변수)를 Y라고 하지만, python에서는 X를 feature, Y를 target이라고 한다.

#### 1) Python for Regression

sklearn을 이용하여 회귀분석을 수행하는 코드에 대해 간단히 알아보도록 한다. 우선 model이 들어있는 package를 import 한다.

```
from sklearn.linear_model import LinearRegression
```

package에서 이용할 모델을 정의한다.

```
model = LinearRegression()
```

정의한 모델에 이용할 수 있는 메서드는 다음과 같다.

Methods	
fit(X, y[, sample_weight])	Fit linear model
get_params([deep])	Get parameters for this estimator
predict(X)	Predict using the linear model
score(X, y[, sample_weight])	Returns the coefficient of determination R-squared of the prediction

linear regression model의 intercept와 coefficient는 아래의 과정을 통해 확인할 수 있다.

```
# intercept
model.intercept_

# coefficient
model.coef_
```

R의 경우 model fitting 후 summary() 함수를 이용할 수 있지만 sklearn은 machine learning에 초점을 맞춘 package이기 때문에 summary 함수를 제공하지 않는다. 대신, statsmodels package에서 summary 함수를 제공한다.

## 2) simple linear regression(SLR)

simple linear regression 은 두 변수(x,y) 사이에 선형성이 존재한다고 가정한다. 따라서, 이 가정을 토대로 linear model 을 추정한다.

$$y = \beta_0 + \beta_1 \times x$$

일반적으로 회귀식(simple linear regression line)을 위와 같이 나타낸다. 회귀분석의 과정은 데이터를 이용하여 위의 식에서  $\beta_0$ (intercept)와  $\beta_1$ (slope)을 추정하는 것이다. 이때, 데이터에 의해 추정된 회귀식은 다음과 같이 나타낸다.

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

따라서,  $\hat{\beta}_0$  와  $\hat{\beta}_1$ 은 각각 estimated intercept 와 estimated slope 이라고 하며,  $\hat{y}$ 는 the fitted value/predicted value 라고 한다. 여기서 추정된  $\hat{y}$ 과 실제  $y$ 값으로부터 잔차(residual,  $\hat{u}_i$ )의 개념이 발생하는데, 잔차( $\hat{u}_i$ )란 실제  $y$ 와  $\hat{y}_i$ 의 차이로 다음과 같이 구할 수 있다.

$$\hat{u}_i = Y_i - \hat{Y}_i$$

회귀식을 추정하기 위해선 잔차를 제곱하여 그 합을 최소화하는  $\beta_0$ 와  $\beta_1$ 을 선택한다. 잔차의 제곱합인 residual sum of squares(RSS)의 값을 최소화하는  $\beta_0$ 와  $\beta_1$ 를 추정하는 식은 다음과 같다.

$$RSS = \sum_{i=1}^N (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^N (Y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

마지막으로 추정한 회귀식이 예측값을 잘 맞추는지 확인하기 위해선  $y$  와  $\hat{y}$ 에 대한 그래프를 통해 확인한다. 예시 데이터를 이용하여 python 에서 simple linear regression 을 추정하는 과정을 알아보자.

### cf. OLS(Ordinary Least Squares), 최소제곱법

잔차제곱합(RSS)을 최소화하는 이러한 방법을 최소제곱법(OLS)하고 한다.

$$\operatorname{argmin}_{\beta_0, \beta_1} RSS = \operatorname{argmin}_{\beta_0, \beta_1} \sum_{i=1}^N (Y_i - \hat{Y}_i)^2 = \operatorname{argmin}_{\beta_0, \beta_1} \sum_{i=1}^N (Y_i - \beta_0 - \beta_1 x_i)^2$$

최소제곱법은 계수 계산을 위해 다음과 같이 간단한 공식을 이용한다.

$$\beta_1 = \frac{\sum_{i=1}^N (Y_i - \bar{Y})(X_i - \bar{X})}{\sum (X_i - \bar{X})^2}$$

$$\beta_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

역사적으로, OLS가 회귀분석에 널리 쓰이게 된 이유 중 하나가 계산의 편의성 때문이다. 계산속도는 빅 데이터 시대에 중요한 요소 중 하나이다.

### a) import the dataset

회귀분석을 위한 라이브러리를 불러들인다. (seaborn과 matplotlib는 visualization을 위한 라이브러리이다.)

```
# import Library
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Library for regression
from sklearn.linear_model import LinearRegression
from sklearn import datasets

# 그래프 배경화면을 격자 화색으로 설정
sns.set_theme(color_codes = True)
```

경력(YearsExperience)과 월급(Salary)에 관한 data를 import한다. 경력(YearsExperience)은 x(설명변수)이며, 월급(Salary)은 y(반응변수)이다.

```
# Load dataset
mydata = pd.read_csv("https://raw.githubusercontent.com/sudarshan-koirala/Salary-Prediction-based-on-Years-of-Experience/master/Salary_Data.csv")
mydata.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

NOTE)

데이터 주소

[https://raw.githubusercontent.com/sudarshan-koirala/Salary-Prediction-based-on-Years-of-Experience/master/Salary\\_Data.csv](https://raw.githubusercontent.com/sudarshan-koirala/Salary-Prediction-based-on-Years-of-Experience/master/Salary_Data.csv)

### b) data preprocessing

model에 데이터를 fitting하기 위해서 전처리 과정이 필요하다. 데이터를 x와 y로 나눈다.

```
X = mydata.iloc[:, :-1].values
Y = mydata.iloc[:, 1].values
```

### c) model fitting

다음으로 simple regression model에 위의 데이터를 fitting시킨다. 우선, sklearn의 linear\_model library에서 LinearRegression class를 import하여 model을 불러온다.

```
# model fitting
from sklearn.linear_model import LinearRegression
mymodel = LinearRegression()
mymodel.fit(X, Y)
```

```
LinearRegression()

# result
print(f'intercept : {mymodel.intercept_}')
print(f'slope : {mymodel.coef_[0]}')


intercept : 25792.200198668696
slope : 9449.962321455076
```

NOTE)

sklearn의 linear regression은 OLS 방법을 이용하여 model fitting을 한다.

#### d) Test model

다음으로 추정된 model이 좋은 성능을 보이는지 판단해 본다. 먼저, 데이터와 추정된 model을 시각화하여 그림을 통해 관찰한다.

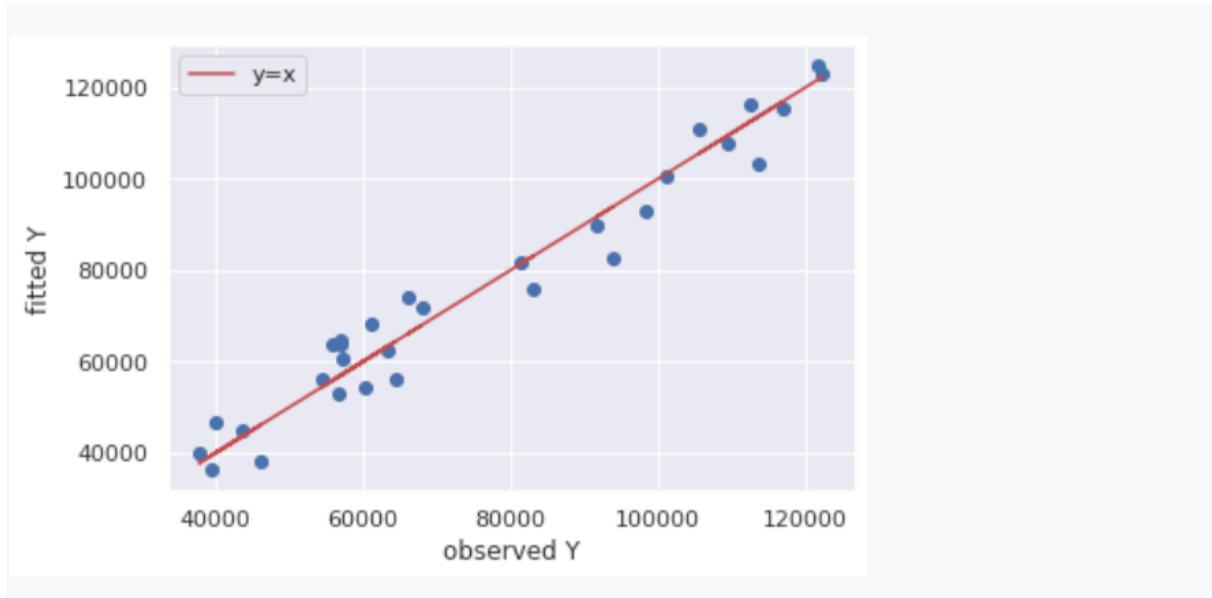
```
# Plotting the observation
plt.scatter(X, Y, color = "b")

# Plotting the regression line
plt.plot(X, mymodel.predict(X), color = "r")      # R의 Line 함수와 동일
plt.title("Salary vs Experience")
plt.xlabel("Years of Experience")
plt.ylabel("Salaries")
plt.show()
```



fitting된 model이 observation들을 지나가고 있는 것을 확인할 수 있다.  $y_i$ 와  $\hat{y}_i$ 에 대한 plot으로 다시 한번 확인한다.

```
# Plotting the observed and fitted value
plt.scatter(Y, mymodel.predict(X), color = "b")
plt.plot(Y, Y, color = "r", label = "y=x")    # y=x line
plt.xlabel("observed Y")
plt.ylabel("fitted Y")
plt.legend()
plt.show()
```



$Y_i$ 와  $\hat{Y}_i$ 가  $y = x$  위에 분포되어 있으므로 model이 예측값을 잘 추정하고 있음을 확인할 수 있다. 다음으로,  $R^2$ 를 이용하여 model의 성능을 판단해본다. fitting하는 model의 성능 지표를 어떤 방법을 이용하여 반환하는지 알기 위해선 help를 이용한다.

```
help(LinearRegression.score)

Help on function score in module sklearn.base:

score(self, X, y, sample_weight=None)
    Return the coefficient of determination of the prediction.

    The coefficient of determination :math:`R^2` is defined as
    :math:`(1 - \frac{\sum u}{\sum v})`, where :math:`u` is the residual
    sum of squares ``((y_true - y_pred)** 2).sum()`` and :math:`v`
    is the total sum of squares ``((y_true - y_true.mean()) ** 2).sum()``.
    The best possible score is 1.0 and it can be negative (because the
    model can be arbitrarily worse). A constant model that always predicts
    the expected value of `y`, disregarding the input features, would get
    a :math:`R^2` score of 0.0.

Parameters
-----
X : array-like of shape (n_samples, n_features)
    Test samples. For some estimators this may be a precomputed
    kernel matrix or a list of generic objects instead with shape
    ``(``n_samples, n_samples_fitted``)``, where ``n_samples_fitted``
    is the number of samples used in the fitting for the estimator.

y : array-like of shape (n_samples,) or (n_samples, n_outputs)
    True values for `X`.

sample_weight : array-like of shape (n_samples,), default=None
    Sample weights.

Returns
-----
score : float
```

```
:math:`R^2` of ``self.predict(X)`` wrt. `y`.
```

#### Notes

-----

The :math:`R^2` score used when calling ``score`` on a regressor uses ``multioutput='uniform\_average'`` from version 0.23 to keep consistent with default value of :func:`~sklearn.metrics.r2\_score`. This influences the ``score`` method of all the multioutput regressors (except for :class:`~sklearn.multioutput.MultiOutputRegressor`).

```
print(round(mymodel.score(X, Y), 3))
```

```
0.957
```

$R^2$ 의 값이 0.957로 매우 높다. 이는 모델이  $Y$ 의 변동량의 대부분을 설명한다는 것을 의미한다.

### cf. 모델 평가

모델의 성능을 판단하기 위한 여러가지 지표들이 있다.

#### a) RMSE( Root Mean Square Error, 평균 제곱근 오차)

RMSE는 예측된  $\hat{y}_i$ 값들의 평균 제곱 오차의 제곱근을 말한다.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

RMSE는 전반적인 모델의 정확도를 측정하고 다른 모델과 비교하기 위한 기준이 된다.

#### b) $R^2$ ( R-squared, 결정계수)

또 다른 유용한 지표는 R-squared( $R^2$ )이다.  $R^2$ 의 범위는 0에서 1까지이며 모델 데이터의 변동률을 측정한다. 모델이 데이터에 얼마나 적합한지 평가하고자 할 때, 회귀분석을 설명하기 위한 용도로 활용된다.  $R^2$ 을 구하는 공식 다음과 같다.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

분모는 Y의 분산에 비례한다. 또한, 모델에 대한 예측값이 실제값과 일치하면  $R^2$ 은 1이 된다. 잔차는 다음과 같이 계산된다.

$$residuals = y - \hat{y}$$

이 식을 아래와 같이 변형하면,

$$y = \hat{y} + residuals$$

$R^2$ 의 분모는 다음과 같이 분해된다.

$$\sum_{i=1}^n (y - \bar{y})^2 = \sum_{i=1}^n (\hat{y} - \bar{y})^2 + \sum_{i=1}^n residual^2$$

위의 식은 모델로 설명 가능한 부분인  $\sum_{i=1}^n (\hat{y} - \bar{y})^2$ 과 모델로 설명하지 못하는 부분인 잔차제곱합으로 분해된다는 것을 의미한다. 따라서,  $R^2$ 는 모델이 데이터를 얼마나 잘 설명하고 있는지에 대한 지표라고 생각할 수 있다.

## ● statsmodels

statsmodels package의 summary 함수의 결과와 비교한다.

```
import statsmodels.api as sm
X = sm.add_constant(X) # intercept 항 추가
model = sm.OLS(Y, X).fit()
model.summary()
```

OLS Regression Results

<b>Dep. Variable:</b>	y	<b>R-squared:</b>	0.957			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.955			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	622.5			
<b>Date:</b>	Wed, 01 Feb 2023	<b>Prob (F-statistic):</b>	1.14e-20			
<b>Time:</b>	07:48:48	<b>Log-Likelihood:</b>	-301.44			
<b>No. Observations:</b>	30	<b>AIC:</b>	606.9			
<b>Df Residuals:</b>	28	<b>BIC:</b>	609.7			
<b>Df Model:</b>	1					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	2.579e+04	2273.053	11.347	0.000	2.11e+04	3.04e+04
x1	9449.9623	378.755	24.950	0.000	8674.119	1.02e+04
<b>Omnibus:</b> 2.140 <b>Durbin-Watson:</b> 1.648						
<b>Prob(Omnibus):</b> 0.343 <b>Jarque-Bera (JB):</b> 1.569						
<b>Skew:</b> 0.363 <b>Prob(JB):</b> 0.456						
<b>Kurtosis:</b> 2.147		<b>Cond. No.</b> 13.2				

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

다음으로 분석에 필요한 파라이터를 빼내는 방법은 다음과 같다.

```
# extract coefficient
print("====")
print("model parameters")
print("====")
print(model.params)
print("====")

# extract t-value
print("t-values")
print("====")
print(model.tvalues)

=====
model parameters
=====
[25792.20019867  9449.96232146]
=====
R-square
=====
0.9569566641435086
=====
t-values
=====
[11.34693968 24.95009424]
```

### 3) multiple linear regression(MLR)

multiple linear regression(MLR)은 multiple regression으로 불리기도 하며, 한 개의 반응변수(Y)를 예측하기 위해 여러개의 설명변수(X)를 이용하는 방법이다. 설명변수가 여러개인 수식은 다음과 같은 형태가 된다.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + e$$

MLR은 직선의 형태가 아니지만, 각 계수( $\beta_i$ )와 그 변수( $X_i$ )들 사이의 관계는 여전히 선형이므로 선형모형이다.

최소제곱법을 이용한 fitting, fitted value, residual의 정의 같은 SLR에서 다른 기타 모든 개념은 MLR에도 그대로 확장되어 적용된다. 예를 들어, fitted value는 다음과 같다.

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_{1,i} X_{1,i} + \hat{\beta}_{2,i} X_{2,i} + \cdots + \hat{\beta}_{p,i} X_{p,i}$$

#### a) import the dataset

diabetes(당뇨병) 데이터를 이용하여 MLR의 회귀식을 추정해보도록 한다. sklearn의 built-in dataset인 당뇨병 환자 데이터를 불러온 후 어떤 Key가 있는지 확인한다.

```
# import package
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Library for regression
from sklearn.linear_model import LinearRegression
from sklearn import datasets

# import dataset
mydata = datasets.load_diabetes()
mydata.keys()

dict_keys(['data', 'target', 'frame', 'DESCR', 'feature_names', 'data_filename',
'target_filename', 'data_module'])
```

dataset을 data frame으로 만든다.

```
# featur columns
df = pd.DataFrame(mydata['data'], columns = mydata['feature_names'])

# target column 추가
df['diabetes_score'] = mydata['target']

df.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	diabetes_score
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

NOTE)

age 데이터의 숫자가 실수형인데, 이는 모든 특성이 -0.2 ~ 0.2 사이에 분포하도록 조정했기 때문이다.

이 데이터에서 target은 당뇨병 수치(diabetes\_score)이고 나머지 feature names에 속하는 age, sex, bmi, bp 등은 feature다. 따라서, 이 데이터는 442명의 사람들을 대상으로 당뇨병 수치를 추정하기 위한 10가지 feature를 나열한 것이다.

```
print('target : diabetes_score(당뇨병 수치)')

for i, feature_name in enumerate(mydata.feature_names):
    print(f'feature {i+1} : {feature_name}')

target : diabetes_score(당뇨병 수치)
feature 1 : age
feature 2 : sex
feature 3 : bmi
feature 4 : bp
feature 5 : s1
feature 6 : s2
feature 7 : s3
feature 8 : s4
feature 9 : s5
feature 10 : s6
```

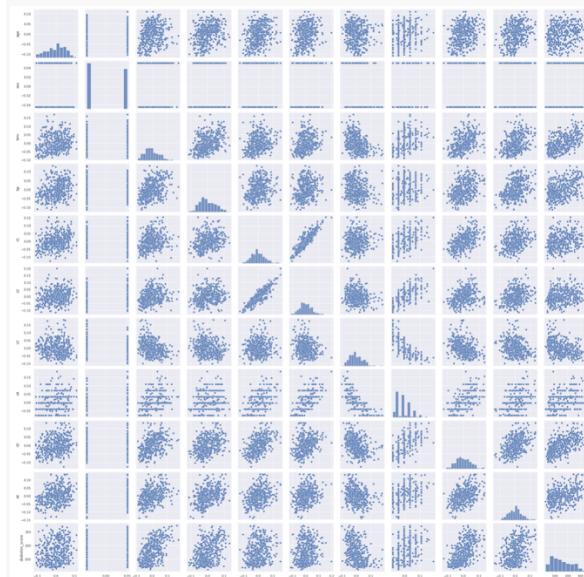
우선, corr()과 pairplot()을 이용해 target 과 변수들 간의 correlation 을 관찰한다.

```
df.corr().tail(1)
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	diabetes_score
diabetes_score	0.187889	0.043062	0.58645	0.441484	0.212022	0.174054	-0.394789	0.430453	0.565883	0.382483	1.0

target과 correlation이 가장 높은 변수는 bmi이다.

```
sns.pairplot(data=df)
plt.show()
```



scatter plot을 통해 sex는 categorical variable임을 알 수 있으며, s1과 s2사이에는 강한 선형성이 관찰된다.

### b) data preprocessing

model fitting을 위해 데이터를 x와 y로 나눈다.

```
X = df[['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']]
Y = df[['diabetes_score']]
```

### c) model fitting

MLR에 위의 데이터를 fitting 한다.

```
# model fitting
from sklearn.linear_model import LinearRegression
mymodel = LinearRegression()
mymodel.fit(X, Y)

LinearRegression()

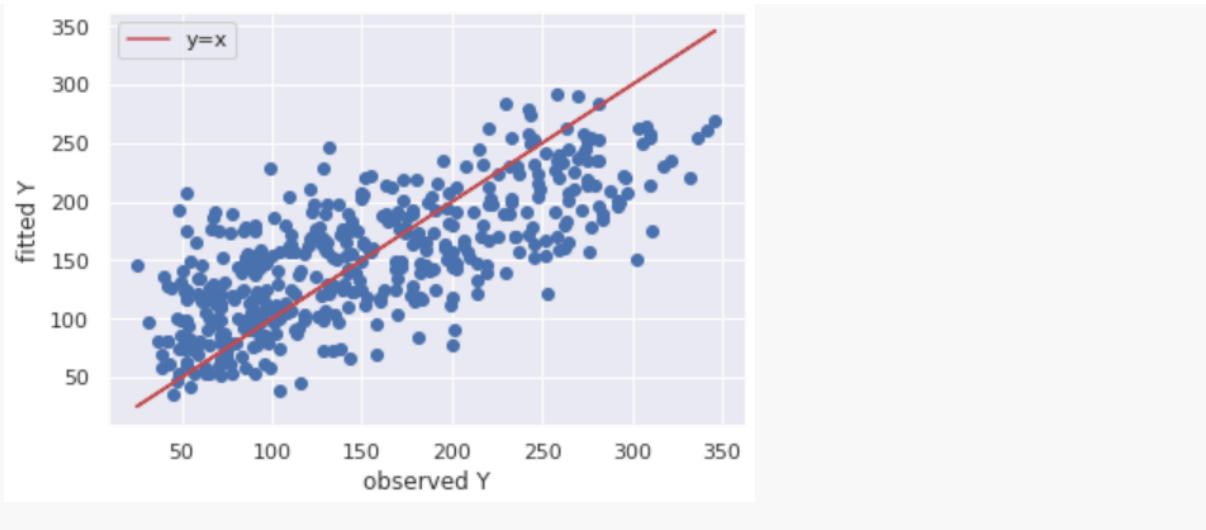
# result
print(f'intercept : {mymodel.intercept_}')
for i, feature_name in enumerate(mydata.feature_names):
    print(f'{feature_name} coef : {mymodel.coef_[0,i]}')

intercept : [152.13348416]
age coef : -10.01219781747065
sex coef : -239.81908936565608
bmi coef : 519.8397867901341
bp coef : 324.3904276893763
s1 coef : -792.1841616283054
s2 coef : 476.74583782366267
s3 coef : 101.04457032134462
s4 coef : 177.06417623225033
s5 coef : 751.2793210873947
s6 coef : 67.62538639104375
```

### d) Test model

마지막으로 시각화와  $R^2$ 값을 이용하여 model의 성능을 판단한다. 우선  $Y_i$ 와  $\hat{Y}_i$ 를 이용하여 plot을 그린다.

```
# Plotting the observed and fitted value
plt.scatter(Y, mymodel.predict(X), color = "b")
plt.plot(Y, Y, color = "r", label = "y=x")
plt.xlabel("observed Y")
plt.ylabel("fitted Y")
plt.legend()
plt.show()
```



$y_i$ 와  $\hat{y}_i$ 가  $y = x$  주변에 가까이 분포되어 있다. 다음으로  $R^2$ 을 이용하여 model의 성능을 판단한다.

```
print(round(mymodel.score(X,Y), 3))
```

```
0.518
```

### cf. statsmodel

statsmodel package의 summary 함수의 결과와 비교한다.

```
import statsmodels.api as sm
X = sm.add_constant(X) # intercept 항 추가
model = sm.OLS(Y, X).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	diabetes_score	R-squared:	0.518			
Model:	OLS	Adj. R-squared:	0.507			
Method:	Least Squares	F-statistic:	46.27			
Date:	Wed, 01 Feb 2023	Prob (F-statistic):	3.83e-62			
Time:	07:49:08	Log-Likelihood:	-2386.0			
No. Observations:	442	AIC:	4794.			
Df Residuals:	431	BIC:	4839.			
Df Model:	10					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	152.1335	2.576	59.061	0.000	147.071	157.196
age	-10.0122	59.749	-0.168	0.867	-127.448	107.424
sex	-239.8191	61.222	-3.917	0.000	-360.151	-119.488
bmi	519.8398	66.534	7.813	0.000	389.069	650.610
bp	324.3904	65.422	4.958	0.000	195.805	452.976
s1	-792.1842	416.684	-1.901	0.058	-1611.169	26.801
s2	476.7458	339.035	1.406	0.160	-189.621	1143.113
s3	101.0446	212.533	0.475	0.635	-316.685	518.774
s4	177.0642	161.476	1.097	0.273	-140.313	494.442
s5	751.2793	171.902	4.370	0.000	413.409	1089.150
s6	67.6254	65.984	1.025	0.306	-62.065	197.316
				Omnibus: 1.506	Durbin-Watson: 2.029	
				Prob(Omnibus): 0.471	Jarque-Bera (JB): 1.404	
				Skew: 0.017	Prob(JB): 0.496	
				Kurtosis: 2.726	Cond. No. 227.	

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```

# extract coefficient
print("====")
print("model parameters")
print("====")
print(model.params)
print("====")

# extract R-squre
print("R-squre")
print("====")
print(model.rsquared)
print("====")

# extact t-vlaue
print("t-values")
print("====")
print(model.tvalues)

=====
model parameters
=====
const      152.133484
age        -10.012198
sex       -239.819089
bmi        519.839787
bp         324.390428
s1        -792.184162
s2         476.745838
s3        101.044570
s4        177.064176
s5        751.279321
s6         67.625386
dtype: float64
=====
R-squre
=====
0.5177494254132934
=====
t-values
=====
const      59.061427
age        -0.167570
sex        -3.917184
bmi        7.813196
bp         4.958435
s1        -1.901163
s2         1.406187
s3         0.475431
s4         1.096538
s5         4.370394
s6         1.024872
dtype: float64

```

## 4. Classification

$y$ 가 범주형 자료인 경우 설명변수를 이용하여  $y$ 를 잘 분류할 수 있는 classifier를 잘 구축하는 것이 목표이다. Regression과의 차이점은 classification의 경우 OLS 방법을 이용할 수 없기 때문에 loss function을 이용하여 model fitting을 다른 방식으로 진행한다. 여기서는 Logistic Regression과 LDA를 이용한 classification을 설명한다.

### 1) logistic regression

일반적인 회귀분석은 독립변수들에 의해 종속변수의 변화가 직선적으로 변한다고 가정한다. 이런 가정은 종속변수가 이항적인 상황에서 적합하지 않다. 종속변수가 범주형 변수인 상황에서 적용할 수 있는 회귀분석 방법이 로지스틱 회귀분석이며 예측값으로 확률을 계산하며 각 클래스에 속할 확률을 계산한다.

Terminology	
로짓(logit)	(0~1이 아니라) 실수 전체의 범위에서 어떤 클래스에 속할 확률을 결정하는 함수
오즈(odds)	'실패(0)'에 대한 '성공(1)'의 비율
연결 함수 (log odds, logit function)	계산 결과가 확률로 나오기 위해 로짓의 값을 입력받아 확률로 변환해주는 함수

logistic regression의 확률  $p$ 는 데이터의 정보를 반영하여 target이 '1'이 될 확률로 정의한다.

$$p = P(Y = 1|x)$$

우선, logistic regression의 가정은 다음과 같다. 확률에 대한 log odds는 아래의 식으로 정의한다.

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

따라서, log odds가  $x$ 의 linear model이다. 이때,  $p$ 는 확률이므로 0과 1사이의 값이 나와야하며, 우변을 아래와 같이 정의하면

$$X\beta = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

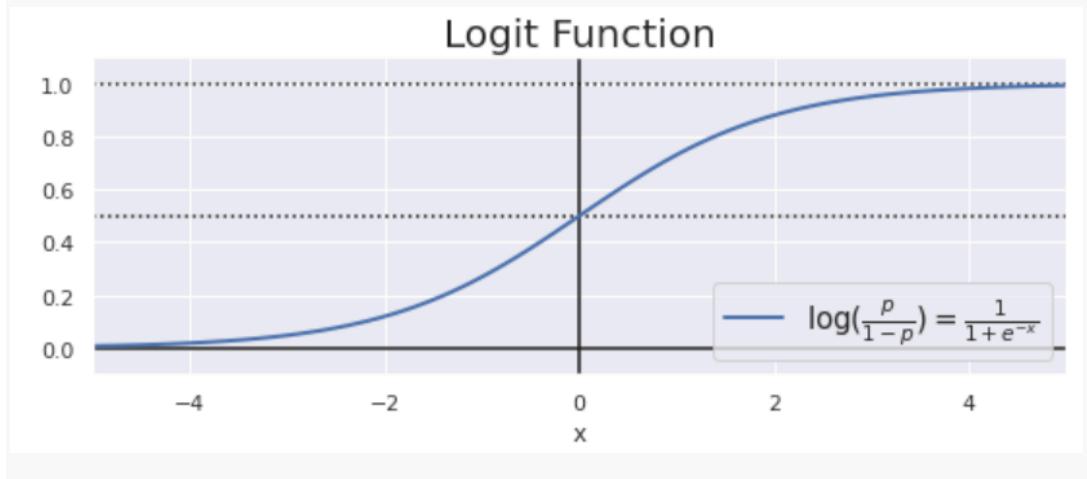
$-\infty < X\beta < \infty$ 와 같은 범위를 가지게 된다.

즉,  $0 \leq p \leq 1$ 와  $-\infty < X\beta < \infty$ 의 범위를 연결하기 위해서 link function이 필요하며, logistic regression에서 link function은 logit function이다.

$$\text{logit}(p) = \log \frac{p}{1-p}$$

```
# plotting Logit function
x = np.linspace(-5, 5, 100)
odds = 1 / (1 + np.exp(-x))
plt.figure(figsize=(9, 3))
plt.plot([-5, 5], [0, 0], "k-")
plt.plot([-5, 5], [0.5, 0.5], "k:")
plt.plot([-5, 5], [1, 1], "k:")
plt.plot([0, 0], [-1.1, 1.1], "k-")
plt.plot(x, odds, "b-", linewidth=2, label=r"\log(\frac{p}{1-p}) = \frac{1}{1 + e^{-x}}")
plt.title("Logit Function", fontsize = 20)
plt.xlabel("x")
plt.legend(loc="lower right", fontsize=15)
```

```
plt.axis([-5, 5, -0.1, 1.1])
plt.show()
```



위의 그림에서 알 수 있듯  $\frac{1}{1+e^{-x}}$  transformation은  $x$ 의 값이  $-\infty$ 로 가면 0의 값으로,  $x$ 의 값이  $\infty$ 로 가면 1의 값으로 출력되도록 한다.

따라서, log odds에서

$$\log \frac{p}{1-p} \rightarrow \infty \text{ when } p \rightarrow 1$$

$$\log \frac{p}{1-p} \rightarrow -\infty \text{ when } p \rightarrow 0$$

위와 같은 관계가 link function(logit function)으로 매칭된다. 그렇다면,  $p$ 는 어떻게 계산하는지에 대해 알아보도록 한다.

$$\begin{aligned} \log \frac{p}{1-p} &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = X\beta \\ \Leftrightarrow \frac{p}{1-p} &= e^{X\beta} \\ \Leftrightarrow p &= (1-p) e^{X\beta} \\ \Leftrightarrow p &= e^{X\beta} - p e^{X\beta} \\ \Leftrightarrow p(1 + e^{X\beta}) &= e^{X\beta} \\ \therefore p &= \frac{e^{X\beta}}{1 + e^{X\beta}} \end{aligned}$$

따라서,  $\beta$  을 추정하면  $p$ 를 계산할 수 있다.

다음으로 logistic regression의 예시를 보도록 한다.

데이터 분석의 목적은 학생들의 GPA 시험성적을 이용한 합격 여부 예측이다.

### a) import dataset

```
# import package
import pandas as pd
math = pd.read_csv("http://home.ewha.ac.kr/~josong/dm/mathcat.data", sep = '\s+')
math.head()
```

	hsgpa	hsengl	hscalc	course	passed	outcome
1	78.0	80	Yes	Mainstrm	No	Failed
2	66.0	75	Yes	Mainstrm	Yes	Passed
3	80.2	70	Yes	Mainstrm	Yes	Passed
4	81.7	67	Yes	Mainstrm	Yes	Passed
5	86.8	80	Yes	Mainstrm	Yes	Passed

### b) data preprocessing

```
# 필요없는 변수 제거
math = math.drop("outcome", axis = 1)
```

hscalc, course, passed 는 범주형 변수(binary categorical variable)로 python에서는 라벨 인코딩과 원-핫 인코딩이 필요하다. 각 변수의 범주의 빈도수를 확인한다.

```
math['hscalc'].value_counts()
```

```
Yes      373
No       21
Name: hscalc, dtype: int64
```

```
math['passed'].value_counts()
```

```
Yes     236
No     158
Name: passed, dtype: int64
```

```
math['course'].value_counts()
```

```
Mainstrm    328
Catch-up     35
Elite        31
Name: course, dtype: int64
```

Y(target)에 해당되는 passed는 라벨 인코딩을 한다.

```
# 라벨 인코딩
from sklearn.preprocessing import LabelEncoder
items = math.columns[4::2]
le = LabelEncoder()

for i in items:
    math[i] = le.fit_transform(math[i])

math.head()
```

	hsgpa	hsengl	hscalc	course	passed
1	78.0	80	Yes	Mainstrm	0
2	66.0	75	Yes	Mainstrm	1
3	80.2	70	Yes	Mainstrm	1
4	81.7	67	Yes	Mainstrm	1
5	86.8	80	Yes	Mainstrm	1

```
mymath1 = math.copy()
mymath2 = math.copy()
```

hscalc와 course는 범주형 변수이므로 원-핫 인코딩을 이용한다. 파이썬에서는 범주형 변수에 대한 정의를 하지 않으므로 원-핫 인코딩 과정을 통해 범주형 변수임을 정의한다.

원-핫 인코딩을 하기 위한 두가지 방법이 있다.

#### b-1) pandas의 pd.get\_dummies 함수를 이용

```
import pandas as pd
pd.get_dummies(mymath1['course'])
```

	Catch-up	Elite	Mainstrm
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
5	0	0	1
...	...	...	...
390	0	0	1
391	0	0	1
392	0	0	1
393	0	0	1
394	1	0	0

394 rows × 3 columns

course는 Catch-up, Elite, Mainstrm 3개의 범주로 이루어져 있는 변수이므로, 해당 범주들이 하나의 컬럼으로 생성되었음을 확인할 수 있다. hscalc도 동일한 과정을 적용한다.

```
mymath1 = pd.get_dummies(data = mymath1, columns = ['course', 'hscalc'], prefix = ['course', 'hsclac'])
```

mymath1

	hsgpa	hsengl	passed	course_Catch-up	course_Elite	course_Mainstrm	hsclac_No	hsclac_Yes
1	78.0	80	0	0	0	1	0	1
2	66.0	75	1	0	0	1	0	1
3	80.2	70	1	0	0	1	0	1
4	81.7	67	1	0	0	1	0	1
5	86.8	80	1	0	0	1	0	1
...	...	...	...	...	...	...	...	...
390	88.3	90	1	0	0	1	0	1
391	77.0	79	1	0	0	1	0	1
392	80.7	70	1	0	0	1	0	1
393	80.7	81	1	0	0	1	0	1
394	82.2	86	1	1	0	0	0	1

394 rows × 8 columns

NOTE)

LabelEncoder는 alphabetic order로 라벨 인코딩을 진행한다.

NOTE)

3개 이상의 범주형 변수에 대해서 라벨 인코딩만 할 경우 파이썬은 이 변수를 numeric 변수로 인식하기 때문에 반드시 원-핫 인코딩을 통해 범주형 변수로 인식할 수 있도록 변환해야 한다.

NOTE)

pd.get\_dummies에서 prefix 옵션은 원래 컬럼의 이름을 앞에 붙여주도록 한다.

### b-2) sklearn 의 OneHotEncoder 를 이용

sklearn 의 OneHotEncoder 를 이용하는 경우 벡터의 입력을 허용하지 않기 때문에, reshape 메서드를 통해 array 로 변환 후 입력해야 한다.

```
# One-hot Encoding 을 할 column 추출

# mymath2[['course']]는 벡터로 추출되므로 원핫인코딩 과정에서 에러가 발생한다.
mycourse = mymath2['course']
print(type(mycourse))

# Series -> array
mycourse = mycourse.values.reshape(-1,1)
print(type(mycourse))
print(mycourse.shape)

<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>
(394, 1)
```

```
# OneHotEncoding 모델 정의
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse = False)
```

위의 코드에서 OneHotEncoder를 불러온 뒤 정의하였다. 이때, 옵션에서 sparse=TRUE가 디폴트이며 이 옵션을 Matrix를 반환하다. 그러나, 원핫인코딩에서 데이터 array형으로 다루기 때문에 sparse=False로 지정한다.

```
# One-Hot Encoding
ohe.fit(mycourse)
one_hot_encoded = ohe.transform(mycourse)

# dataframe
ohe_df = pd.DataFrame(one_hot_encoded, columns = ohe.categories_[0])
ohe_df
```

	Catch-up	Elite	Mainstrm
0	0.0	0.0	1.0
1	0.0	0.0	1.0
2	0.0	0.0	1.0
3	0.0	0.0	1.0
4	0.0	0.0	1.0
...	...	...	...
389	0.0	0.0	1.0
390	0.0	0.0	1.0
391	0.0	0.0	1.0
392	0.0	0.0	1.0
393	1.0	0.0	0.0

394 rows × 3 columns

원핫 인코딩 후 만들어진 data frame의 인덱스는 0부터 시작하기 때문에 원래 데이터에 병합 시 인덱스가 맞지 않아 NA 데이터가 만들어진다. 따라서, ohe\_df의 index가 1부터 시작하도록 만든다.

```
# mymath2 와 ohe_df 의 index 가 다르므로 조정
ohe_df.index = ohe_df.index + 1

# 데이터 병합
mydf = pd.concat([mymath2, ohe_df], axis = 1)
mydf.head(3)
```

	hsgpa	hsengl	hscalc	course	passed	Catch-up	Elite	Mainstrm
1	78.0	80	Yes	Mainstrm	0	0.0	0.0	1.0
2	66.0	75	Yes	Mainstrm	1	0.0	0.0	1.0
3	80.2	70	Yes	Mainstrm	1	0.0	0.0	1.0

또 다른 범주형 변수인 hscalc도 원핫 인코딩을 한다.

```
# hscalc 을 one-hot encoding
myhscalc = mymath2['hscalc']
myhscalc = myhscalc.values.reshape(-1,1)

ohe.fit(myhscalc)
one_hot_encoded = ohe.transform(myhscalc)

ohe_df = pd.DataFrame(one_hot_encoded, columns = ohe.categories_[0])

ohe_df.index = ohe_df.index + 1

# 데이터 병합
mydf = pd.concat([mydf, ohe_df], axis = 1)
mydf
```

	hsgpa	hsengl	hscalc	course	passed	Catch-up	Elite	Mainstrm	No	Yes
1	78.0	80	Yes	Mainstrm	0	0.0	0.0	1.0	0.0	1.0
2	66.0	75	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
3	80.2	70	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
4	81.7	67	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
5	86.8	80	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...
390	88.3	90	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
391	77.0	79	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
392	80.7	70	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
393	80.7	81	Yes	Mainstrm	1	0.0	0.0	1.0	0.0	1.0
394	82.2	86	Yes	Catch-up	1	1.0	0.0	0.0	0.0	1.0

394 rows × 10 columns

기존의 hscalc와 course는 제거한다.

mydf.drop(["hscalc", "course"], axis = 1)

	hsgpa	hsengl	passed	Catch-up	Elite	Mainstrm	No	Yes
1	78.0	80	0	0.0	0.0	1.0	0.0	1.0
2	66.0	75	1	0.0	0.0	1.0	0.0	1.0
3	80.2	70	1	0.0	0.0	1.0	0.0	1.0
4	81.7	67	1	0.0	0.0	1.0	0.0	1.0
5	86.8	80	1	0.0	0.0	1.0	0.0	1.0
...	...	...	...	...	...	...	...	...
390	88.3	90	1	0.0	0.0	1.0	0.0	1.0
391	77.0	79	1	0.0	0.0	1.0	0.0	1.0
392	80.7	70	1	0.0	0.0	1.0	0.0	1.0
393	80.7	81	1	0.0	0.0	1.0	0.0	1.0
394	82.2	86	1	1.0	0.0	0.0	0.0	1.0

394 rows × 8 columns

데이터를 x, y로 나눈다.

```

Y = mymath1.iloc[:, 2]
mymath1.pop("passed")
X = mymath1.iloc[:, :]

```

### c) model fitting

```

from sklearn.linear_model import LogisticRegression

import warnings
warnings.filterwarnings(action = 'ignore')

mymodel = LogisticRegression()
mymodel.fit(X, Y)

LogisticRegression()

```

#### d) Test model

학습된 모델의 예측값과 실제값을 confusion matrix를 이용해 판단한다.

```
# confusion matrix
from sklearn.metrics import confusion_matrix
mymtx = confusion_matrix(mymodel.predict(X), Y)
mymtx

array([[ 98,  38],
       [ 60, 198]])

# misclass rate
misclass_rate = (mymtx[0,1] + mymtx[1,0])/(sum(sum(mymtx)))
misclass_rate

0.24873096446700507
```

99(No)명과 198(Yes)명에 대해서는 올바르게 추정하였다. 그러나 60명과 38명에 대해서는 잘못 분류하고 있으므로 경계 확률을 변화시키면서 오분류율이 작아지도록 더 좋은 추정결과를 찾아보도록 한다.

다음은 분류 경계 확률을 0.4~0.6 사이에서 0.001 씩 증가시켜 오분류율을 최소화하는 경계 확률과 그때의 오분류율을 구하는 코드이다.

```
# range()는 float 형이 들어갈 수 없으므로 float 를 삽입할 수 있는 range 함수를 정의
def range_with_floats(start, stop, step):
    while stop > start:
        yield start
        start += step

# class 1 // 속할 확률, [:,0]인 경우 class 0 // 속할 확률
prob = mymodel.predict_proba(X)[:,1]

# 값들을 저장할 list 와 dictionary
mylist = []
res = {}

n = len(math)

# 최소의 오분류율을 만드는 경계확률을 찾기 위한 반복문
for i in range_with_floats(0.4, 0.601, 0.001):
    for j in range(0, n):

        # 경계확률보다 크면 class 1로 분류, 아니면 0으로 분류
        if prob[j] >= i:
            mylist.append(1)
        else:
            mylist.append(0)

    # 오분류율 구하기 위해 필요한 confusion matrix
    mymtx = confusion_matrix(mylist, Y)

    # 오분류율 계산
    missrate = (mymtx[0,1] + mymtx[1,0])/(sum(sum(mymtx)))
```

```
# dictionary // 경계확률을 key로 오분류율을 value로 저장
res[round(i,3)] = round(missrate,4)

# List 초기화
mylist = []

print(f'최소의 오분류율은 {min(res.values())} 이며 이때의 경계확률은 {min(res, key = res.get)} 이다.')

```

최소의 오분류율은 0.2437이며 이때의 경계확률은 0.512이다.

NOTE)

오분류율 계산하는 함수는 sklearn에 내장되어 있으며 뒤에서는 이 함수를 이용하여 오분류율 계산할 것이다.

### cf. statsmodel

statsmodel package의 summary 함수의 결과와 비교한다.

```
import statsmodels.api as sm

# intercept 항 추가
X = sm.add_constant(X)

# model fitting
model = sm.Logit(Y, X).fit()

# summary
print(model.summary())

Optimization terminated successfully.
    Current function value: 0.534132
    Iterations 8
    Logit Regression Results
=====
Dep. Variable:          passed    No. Observations:             394
Model:                 Logit     Df Residuals:                  388
Method:                MLE      Df Model:                      5
Date:      Wed, 01 Feb 2023   Pseudo R-squ.:            0.2068
Time:      07:49:10           Log-Likelihood:        -210.45
converged:            True    LL-Null:                  -265.33
Covariance Type:    nonrobust   LLR p-value:       4.605e-22
=====

              coef      std err          z      P>|z|      [0.025      0.975]
-----
const      -8.0872        nan        nan        nan        nan        nan
hsgpa       0.2204        0.030      7.336      0.000      0.161      0.279
hsengl     -0.0362        0.018     -2.038      0.042     -0.071     -0.001
course_Catch-up -3.5492  8.7e+06   -4.08e-07     1.000    -1.71e+07    1.71e+07
course_Elite   -1.8691  8.7e+06   -2.15e-07     1.000    -1.71e+07    1.71e+07
course_Mainstrm -2.6688  8.62e+06   -3.1e-07     1.000    -1.69e+07    1.69e+07
hsclac_No      -4.6722        2.347     -1.991      0.047     -9.272     -0.072
hsclac_Yes     -3.4150        2.188     -1.561      0.119     -7.703      0.873
=====
```

```

# extract coefficient
print("====")
print("model parameters")
print("====")
print(model.params)
print("====")

# extract t-value
print("t-values")
print("====")
print(model.tvalues)

=====
model parameters
=====
const           -8.087162
hsgpa          0.220355
hsengl         -0.036188
course_Catch-up -3.549221
course_Elite    -1.869147
course_Mainstrm -2.668803
hsclac_No      -4.672169
hsclac_Yes     -3.414990
dtype: float64
=====
t-values
=====
const           NaN
hsgpa          7.335917e+00
hsengl         -2.037729e+00
course_Catch-up -4.078574e-07
course_Elite    -2.147923e-07
course_Mainstrm -3.096193e-07
hsclac_No      -1.990703e+00
hsclac_Yes     -1.561068e+00
dtype: float64

```

R에서 동일한 분석을 실시하면 다음과 같은 결과를 얻는다.

The screenshot shows the RStudio interface with two panes. The left pane is the 'Source' tab containing R code, and the right pane is the 'Console' tab showing the execution results.

```

1 math = read.table("http://home.ewha.ac.kr/~josong/dm/mathcat.data")
2 math
3 attach(math)
4 math$passed = ifelse(math$passed == "No", 0, 1)
5 model = glm(math$passed ~ hsgpa + hsengl + hscalc + course, family = binomial)
6 summary(model)
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

```

Console output:

```

> summary(model)

Call:
glm(formula = math$passed ~ hsgpa + hsengl + hscalc + course,
     family = binomial)

Deviance Residuals:
    Min      IQ   Median      3Q      Max
-2.5517  -0.9811  0.4059   0.8716   2.2061

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -16.30855  2.16185 -7.544 4.56e-14 ***
hsgpa        0.22036  0.03003  7.337 2.19e-13 ***
hsengl       -0.03619  0.01776 -2.038  0.0416 *
hscalcYes    1.25718  0.67282  1.869  0.0617 .
courseElite   1.68007  0.66593  2.523  0.0116 *
courseMainstrm 0.88042  0.48834  1.803  0.0714 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 530.66  on 393  degrees of freedom
Residual deviance: 420.90  on 388  degrees of freedom
AIC: 432.9

Number of Fisher Scoring iterations: 4

```

R과 동일한 회귀계수가 나오도록 X를 다음과 같이 변경 후 모델의 결과를 본다.

```
X = mymath1[['hsgpa', 'hsengl', 'hsclac_Yes', 'course_Elite', 'course_Mainstrm']]

import statsmodels.api as sm

# intercept 항 추가
X = sm.add_constant(X)

# model fitting
model = sm.Logit(Y, X).fit()

# summary
print(model.summary())

Optimization terminated successfully.
    Current function value: 0.534132
    Iterations 6
    Logit Regression Results
=====
Dep. Variable:      passed   No. Observations:            394
Model:              Logit    Df Residuals:                  388
Method:             MLE     Df Model:                      5
Date:       Wed, 01 Feb 2023 Pseudo R-squ.:           0.2068
Time:          07:49:10   Log-Likelihood:        -210.45
converged:        True    LL-Null:                   -265.33
Covariance Type:  nonrobust LLR p-value:      4.605e-22
=====
          coef      std err      z      P>|z|      [0.025      0.975]
-----
const      -16.3086    2.162    -7.543    0.000    -20.546    -12.071
hsgpa       0.2204    0.030     7.336    0.000      0.161     0.279
hsengl     -0.0362    0.018    -2.038    0.042     -0.071    -0.001
hsclac_Yes  1.2572    0.673     1.868    0.062     -0.062     2.576
course_Elite 1.6801    0.666     2.523    0.012      0.375     2.985
course_Mainstrm 0.8804    0.488     1.803    0.071     -0.077     1.838
=====
```

R과 동일한 결과가 나오는 것을 확인할 수 있다.



#### cf. GLM(generalized linear model)

앞서 유도한 로지스틱 모형에서 target은 1에 대한 로그 오즈 값이었다. 하지만, 실제 관찰한 데이터는 그 오즈 값이 아닌 출력값(target이 0 혹은 1)이다. 따라서 이 결과를 fitting하기 위한 방법이 필요하다. logistic regression은 선형회귀를 확장한 일반화선형모형(GLM, generalized linear model)의 special case이다.

## 2) LDA(Linear Discriminant Analysis)

LDA는 로지스틱 회귀와 비슷한 모델로, 종속변수를 분류하기 위한 모델이다. 따라서, 각 범주에 속할 확률을 예측하는 형태로 분류한다.

LDA에는 두 가지 가정이 필요하다.

- 각 class가 정규분포 형태의 확률분포를 가진다.
- 각 class는 같은 형태의 공분산 구조를 가진다.

위의 가정은 다음과 같이 생각할 수 있다.

우선, 설명변수의 클래스가  $K(k = 1, \dots, K)$  개 존재하며 설명변수의 개수는  $p(p\text{-dim})$ 개라고 가정한다. 위의 가정에 따라,  $P(X|Y = K)$ 가 정규분포를 따른다. 이때, LDA를 이용하여 계산하는 것은  $P(Y = K|X)$ 로 class conditional probability를 구하는 과정이다. 이는 베이즈 정리를 이용하여 계산이 가능하다.

LDA를 이용하여 iris 데이터의 종을 분류해보도록 한다.

### a) import dataset

```
# import package
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# import dataset
myiris = datasets.load_iris()
```

### b) data preprocessing

```
X = myiris.data
Y = myiris.target
target_names = myiris.target_names
```

### c) model fitting

```
# LDA
lda = LinearDiscriminantAnalysis(n_components=2) # n_components = (number of class - 1)
res = lda.fit(X, Y).transform(X)
```

### d) Test model

```
# visualization
plt.figure()
colors = ["navy", "turquoise", "darkorange"]

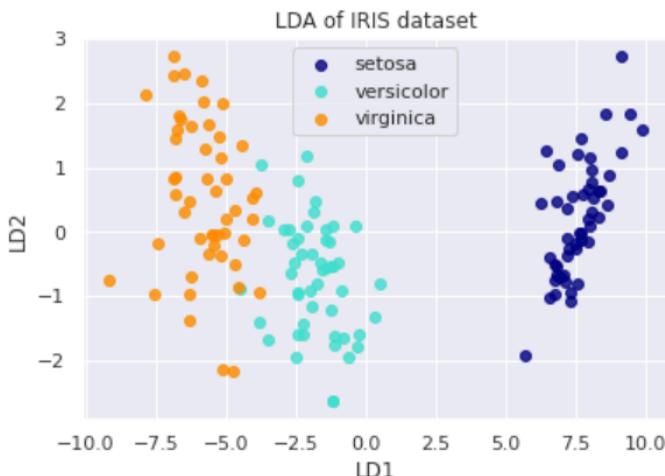
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(res[Y == i, 0], res[Y == i, 1], alpha=0.8, color=color, label=target_name)
```

```

plt.xlabel("LD1")
plt.ylabel("LD2")
plt.legend(loc="best", shadow=False, scatterpoints=1)
# scatterpoints 는 Legend 안의 마커포인트의 수를 입력받음
plt.title("LDA of IRIS dataset")

plt.show()

```



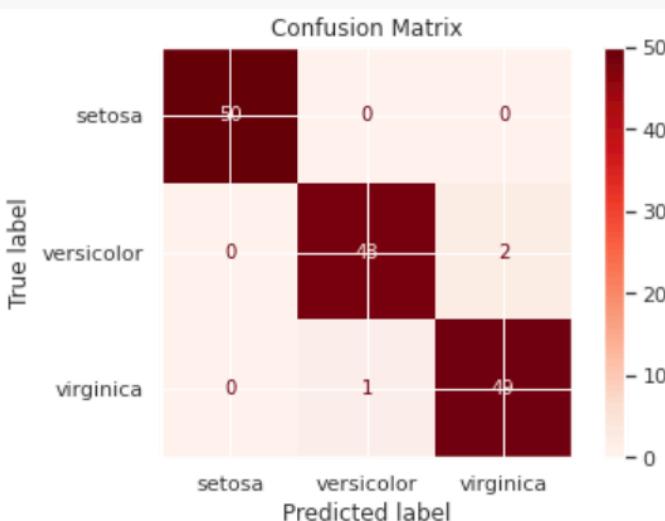
다음으로 confusion matrix를 이용하여 오분류를 관찰한다.

```

# import package
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

# confusion matrix
label = ['setosa', 'versicolor', 'virginica'] # 라벨설정
plot = plot_confusion_matrix(lda,
                             X, Y,
                             display_labels = label,
                             cmap = plt.cm.Reds,
                             normalize = None)
plot.ax_.set_title('Confusion Matrix')
plt.show()

```



```
# confusion matrix
from sklearn.metrics import confusion_matrix
mymtx = confusion_matrix(Y, lda.predict(X))
mymtx

array([[50,  0,  0],
       [ 0, 48,  2],
       [ 0,  1, 49]])
```

다른 종에 비해 setosa의 분류 정확도가 높다는 것을 알 수 있다.

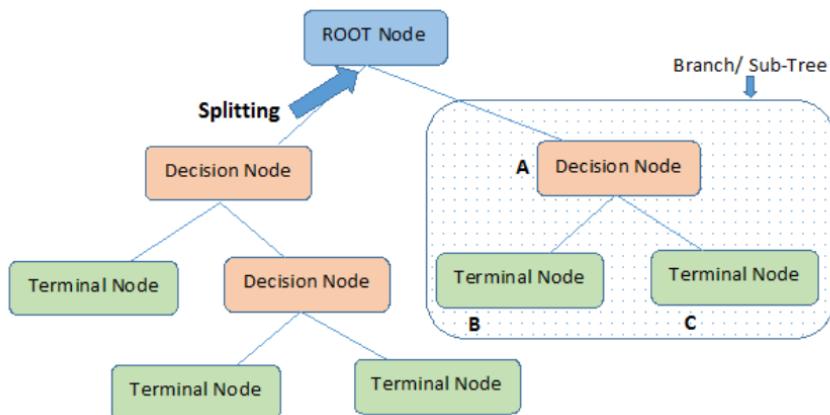
```
import pandas as pd

table = pd.DataFrame(mymtx, columns = label, index = label)
table
```

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	48	2
virginica	0	1	49

## 5. CART(Classification and Regression Tree)

의사결정나무(Decision Tree) 알고리즘은 의사결정규칙(decision rule)을 도표화하여 관심 대상이 되는 집단을 몇 개의 소집단으로 분류(classification)하거나 예측(prediction)을 수행하는 분석 방법이다. 데이터 마이닝 기법 중의 하나인 의사결정나무는 각 자료 내에 존재하는 관계와 규칙을 탐색하고 찾아내어 모형화한다. DT는 반복적으로 분류 규칙을 생성하여 가장 효과적인 분류 규칙 조합을 완성해 가며, 의사 결정 규칙을 나무 구조(Tree Structure)로 나타내어 전체 자료를 몇 개의 소집단으로 분류하거나 예측을 수행한다.



### CART : Recursive binary splits that minimize the impurity of child nodes

CART(Classification and Regression Tree)는 불순도(impurity)를 이용하여 binary split을 수행하는 알고리즘이다. 트리 모델은 쉽게 말해 if-then-else 규칙의 집합이라고 할 수 있다. 또한, CART는 재귀 분할 알고리즘을 사용하는데, 이는 예측변수 값을 기준으로 데이터를 반복적으로 분할해나가는 알고리즘이다. 그러나, 일반적인 DT와 달리 CART는 하나의 규칙으로 2개의 하위 노드만을 반복적으로 만든다.

Terminology	
Recursive partitioning (재귀 분할)	마지막 분할 영역에 해당하는 출력이 최대한 비슷한 결과를 보이도록 데이터를 반복적으로 분할하는 것
split value(분할값)	분할값을 기준으로 예측변수를 그 값보다 작은 영역과 큰 영역으로 분할
node(마디, 노드)	DT와 같은 가지치기 형태로 구성된 규칙들의 집합에서, 분할 규칙의 시각적 표시
leaf(잎)	if-then 규칙의 가장 마지막 부분, tree의 마지막 branch이자 최종적인 분류 규칙
loss(손실)	분류하는 과정에서 발생하는 오분류의 수, 손실이 클수록 불순도가 높다.
impurity(불순도)	데이터를 분할한 집합에서 서로 다른 클래스의 데이터가 얼마나 섞여있는지를 나타내는 척도
pruning(가지치기)	학습이 끝난 트리 모델에서 오버피팅을 줄이기 위해 가지들을 하나씩 잘라내는 과정
Split variable	분할되는 설명변수(feature)

## 1) regression tree

regression tree는 RSS를 가장 작게 만드는 변수를 기준으로 split을 선택하며 tree를 만들어간다. 예를 들어,  $X_1, X_2, X_3$  3개의 변수가 있다고 가정하자.  $X_1$ 에는 총 4개의 데이터가 들어있다.

$$\begin{aligned} Y &= [1, 5, 3, 10] \\ X_1 &= [4, 8, 7, 3] \end{aligned}$$

먼저, 데이터를 데이터 프레임 형식으로 만든다.

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd

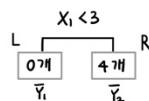
# Data
Y = [1, 5, 3, 10]
X1 = [4, 8, 7, 3]

# Data to DataFrame
df = pd.DataFrame(zip(Y, X1), columns = ['Y', 'X1'])
df
```

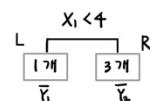
	Y	X1
0	1	4
1	5	8
2	3	7
3	10	3

$X_1$ 에 대하여 split을 만들 수 있는 방법은 4가지가 있다.

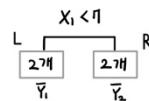
i) Case I



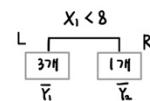
ii) Case II



iii) Case III



iv) Case IV



다음으로 두개의 case에 대한 child node의 RSS를 계산한다. case별 RSS를 구하는 방법은 다음과 같다.

$$RSS = \sum_{i \in L} (Y_i - \bar{Y}_L)^2 + \sum_{i \in R} (Y_i - \bar{Y}_R)^2$$

```

def find_split_node(matrix):

    # import package
    import numpy as np
    import pandas as pd

    # nrow = n, ncol = p
    n, p = matrix.shape

    # node 를 key 로 그때의 RSS 를 value 로 저장하는 dictionary
    res = {}

    for i in range(p-1):
        data = matrix.iloc[:, [0, i+1]]
        data_copy = data.sort_values(by = matrix.columns[i+1], axis = 0)
        col_name = matrix.columns[i+1]

        print('\n')
        print(f'{matrix.columns[i+1]}에서 node 찾기')

        # initialize dictionary
        res1 = {}

        for j in range(n):

            # set node
            node = data_copy.iloc[j,1]
            print(f'node = {node}')

            # set list
            left_list = []
            right_list = []

            # find split node
            for k in range(n):
                if data_copy.iloc[k,1] < node:
                    left_list.append(data_copy.iloc[k,0])
                else:
                    right_list.append(data_copy.iloc[k,0])

            print(left_list, right_list)

            # calculate rss
            left_list = left_list - np.mean(left_list)
            left_list = sum(left_list**2)

            right_list = right_list - np.mean(right_list)
            right_list = sum(right_list**2)

            rss = round(left_list + right_list, 4)

            # res1 // node 와 그때의 rss 를 저장
            res1[node] = rss

        print(res1)

```

```
# X1, X2, X3 에서 rss 를 최소화하는 node 와 rss 를 저장 (nested dictionary 형태)
res[col_name] = {min(res1, key = res1.get) : round(min(res1.values()), 4)}

print('\n')
print(res)

find_split_node(df)

X1에서 node 찾기
node = 3
[] [10, 1, 3, 5]
node = 4
[10] [1, 3, 5]
node = 7
[10, 1] [3, 5]
node = 8
[10, 1, 3] [5]
{3: 44.75, 4: 8.0, 7: 42.5, 8: 44.6667}

{'X1': {4: 8.0}}
```

$X_1$ 에 대하여 RSS를 최소로 만드는 split은  $X_1 = 4$  이므로 이를 저장한 후,  $X_2, X_3$ 에 대해서도 반복적인 (recursive) 계산을 수행한다. 따라서, first split을 선택하기 위한 선택지가 총 3개로 발생하며, 이중 RSS가 가장 최소인 것을 first split으로 선택한다. 다음 split을 결정하기 위해 first split node를 기준으로 데이터를 분류한다. 다음으로, 각각의 분류된 데이터 안에서 다시 동일한 과정을 반복하며, second split node를 구한다.

완성된 regression tree의 마지막 child node(leaf, 잎)에서 예측값들은 그 node에 속한  $y$ 들의 평균이다.

다음으로 전체 데이터에서 regression tree를 이용하여 split node를 찾아보도록 한다. 우선 data를 matrix 형태로 만든다.

```
import warnings
warnings.filterwarnings(action = 'ignore')
import numpy as np
import pandas as pd

Y = [2, 8, 9, 4, 4, 5, 5]
X1 = [1, 5, 3, 10, 6, 2, 5]
X2 = [1, 1, 8, 6, 7, 8, 5]
X3 = [7, 9, 4, 3, 2, 3, 5]

# Data to DataFrame
df = pd.DataFrame((zip(Y, X1, X2, X3)), columns = ['Y', 'X1', 'X2', 'X3'])
df
```

	Y	X1	X2	X3
0	2	1	1	7
1	8	5	1	9
2	9	3	8	4
3	4	10	6	3
4	4	6	7	2
5	5	2	8	3
6	5	5	5	5

```
# first node
find_split_node(df)

X1에서 node 찾기
node = 1
[] [2, 5, 9, 8, 5, 4, 4]
node = 2
[2] [5, 9, 8, 5, 4, 4]
node = 3
[2, 5] [9, 8, 5, 4, 4]
node = 5
[2, 5, 9] [8, 5, 4, 4]
node = 5
[2, 5, 9] [8, 5, 4, 4]
node = 6
[2, 5, 9, 8, 5] [4, 4]
node = 10
[2, 5, 9, 8, 5, 4] [4]
{1: 35.4286, 2: 22.8333, 3: 26.5, 5: 35.4167, 6: 30.8, 10: 33.5}
```

```
X2에서 node 찾기
node = 1
[] [2, 8, 5, 4, 4, 9, 5]
node = 1
[] [2, 8, 5, 4, 4, 9, 5]
node = 5
[2, 8] [5, 4, 4, 9, 5]
node = 6
[2, 8, 5] [4, 4, 9, 5]
node = 7
[2, 8, 5, 4] [4, 9, 5]
node = 8
[2, 8, 5, 4, 4] [9, 5]
node = 8
[2, 8, 5, 4, 4] [9, 5]
{1: 35.4286, 5: 35.2, 6: 35.0, 7: 32.75, 8: 27.2}
```

```
X3에서 node 찾기
node = 2
[] [4, 4, 5, 9, 5, 2, 8]
node = 3
[4] [4, 5, 9, 5, 2, 8]
node = 3
[4] [4, 5, 9, 5, 2, 8]
node = 4
[4, 4, 5] [9, 5, 2, 8]
node = 5
[4, 4, 5, 9] [5, 2, 8]
node = 7
[4, 4, 5, 9, 5] [2, 8]
node = 9
[4, 4, 5, 9, 5, 2] [8]
{2: 35.4286, 3: 33.5, 4: 30.6667, 5: 35.0, 7: 35.2, 9: 26.8333}
```

```
{'X1': {2: 22.8333}, 'X2': {8: 27.2}, 'X3': {9: 26.8333}}
```

최소 rss는 22.8333이며 first node는  $X_1$ 의 2이다. 다음 second node를 찾기 위해선 first node를 기준으로 데이터를 분류한 후 위의 과정을 반복한다.

```
# first node(X1 < 2)를 기준으로 분류
df_copy = df.sort_values(by = "X1", axis = 0)

df_left = df_copy.loc[df['X1'] < 2]
df_right = df_copy.loc[df['X1'] >= 2]
```

df\_left

	Y	X1	X2	X3
0	2	1	1	7

df\_right

	Y	X1	X2	X3
5	5	2	8	3
2	9	3	8	4
1	8	5	1	9
6	5	5	5	5
4	4	6	7	2
3	4	10	6	3

first node를 기준으로 데이터를 df\_left와 df\_right로 나누었다. 다음으로 각 데이터의 second node를 구한다. 이때, df\_left의 경우 데이터의 개수가 1개이므로 node를 구할 수 없다.

```
find_split_node(df_right)

X1에서 node 찾기
node = 2
[] [5, 9, 8, 5, 4, 4]
node = 3
[5] [9, 8, 5, 4, 4]
node = 5
[5, 9] [8, 5, 4, 4]
node = 5
[5, 9] [8, 5, 4, 4]
node = 6
[5, 9, 8, 5] [4, 4]
node = 10
[5, 9, 8, 5, 4] [4]
{2: 22.8333, 3: 22.0, 5: 18.75, 6: 12.75, 10: 18.8}
```

X2에서 node 찾기

```
node = 1
[] [8, 5, 4, 4, 5, 9]
node = 5
[8] [5, 4, 4, 5, 9]
```

```

node = 6
[8, 5] [4, 4, 5, 9]
node = 7
[8, 5, 4] [4, 5, 9]
node = 8
[8, 5, 4, 4] [5, 9]
node = 8
[8, 5, 4, 4] [5, 9]
{1: 22.8333, 5: 17.2, 6: 21.5, 7: 22.6667, 8: 18.75}

```

X3에서 node 찾기

```

node = 2
[] [4, 5, 4, 9, 5, 8]
node = 3
[4] [5, 4, 9, 5, 8]
node = 3
[4] [5, 4, 9, 5, 8]
node = 4
[4, 5, 4] [9, 5, 8]
node = 5
[4, 5, 4, 9] [5, 8]
node = 9
[4, 5, 4, 9, 5] [8]
{2: 22.8333, 3: 18.8, 4: 9.3333, 5: 21.5, 9: 17.2}

```

```
{'X1': {6: 12.75}, 'X2': {5: 17.2}, 'X3': {4: 9.3333}}
```

최소의 rss는 9.3333이며 second node는 X<sub>3</sub>의 4이다.

```

# second node(X3 < 4)를 기준으로 분류
df_copy = df_right.sort_values(by = "X3", axis = 0)

df_left = df_copy.loc[df['X3'] < 4]
df_right = df_copy.loc[df['X3'] >= 4]

```

df\_left

Y	X1	X2	X3
4	4	6	7
5	5	2	8
3	4	10	6

```
print(round(np.mean(df_left.Y), 2))
```

4.33

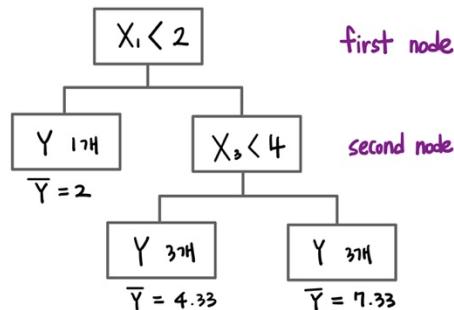
df\_right

Y	X1	X2	X3
2	9	3	8
6	5	5	5
1	8	5	1

```
print(round(np.mean(df_right.Y), 2))
```

7.33

regression tree의 결과는 다음과 같다.



sklearn 의 내장 데이터인 load\_diabetes()를 이용하여 regression tree 를 만들어보도록 한다.

### a) import dataset

```
# import package
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

# import dataset
mydata = datasets.load_diabetes()
mydata.keys()

dict_keys(['data', 'target', 'frame', 'DESCR', 'feature_names', 'data_filename',
'target_filename', 'data_module'])
```

### b) data preprocessing

```
# feature columns
df = pd.DataFrame(mydata['data'], columns = mydata['feature_names'])

# target column 추가
df['diabetes_score'] = mydata['target']

df.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	diabetes_score
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

```
# X, Y split
X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values
```

### c) model fitting

```
from sklearn.tree import DecisionTreeRegressor

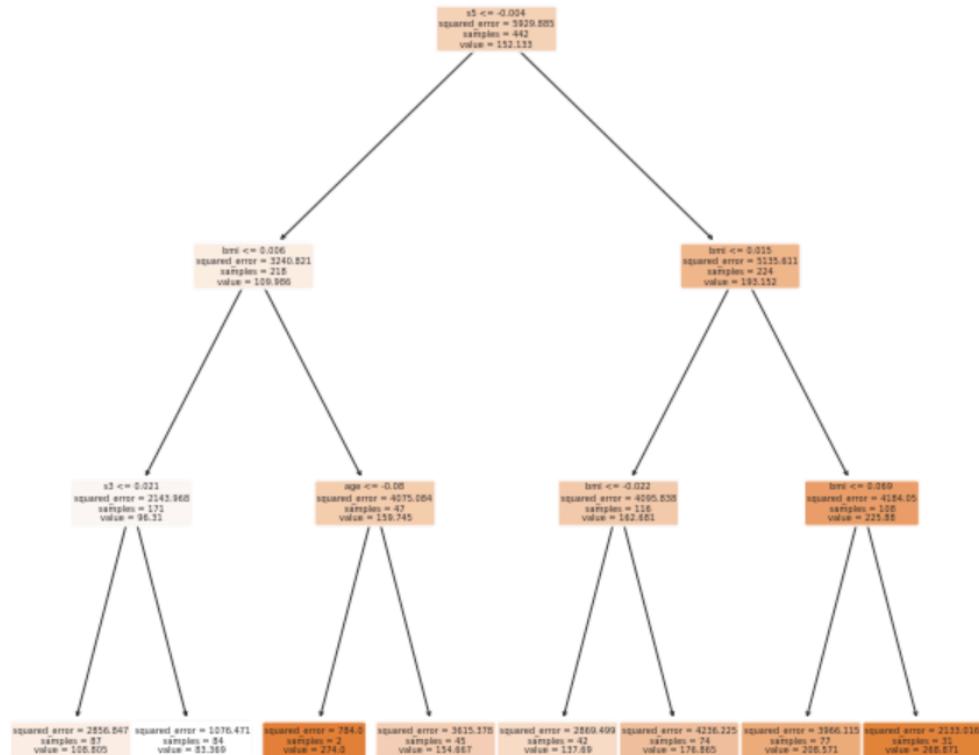
# CART Algorithm for regression
cart_reg = DecisionTreeRegressor(max_depth = 3) # mse
cart_reg.fit(X, Y)

DecisionTreeRegressor(max_depth=3)
```

### d) Test model

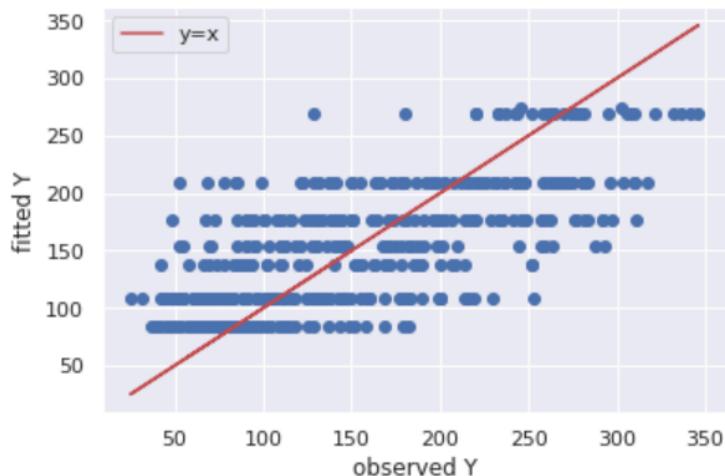
```
from sklearn import tree

# Decision Tree visualization
fig = plt.figure(figsize = (10, 10))
tree.plot_tree(cart_reg,
               feature_names = df.columns[0:10],
               impurity = True,
               rounded = True,
               filled = True
              )
```



```
import matplotlib.pyplot as plt
sns.set_theme(color_codes = True)

plt.scatter(Y, cart_reg.predict(X), color = "b")
plt.plot(Y, Y, color = "r", label = "y=x")
plt.xlabel("observed Y")
plt.ylabel("fitted Y")
plt.legend()
plt.show()
```



```
from sklearn.metrics import r2_score
print(round(r2_score(Y, cart_reg.predict(X)),3))

0.501
```

## 2) classification tree

classification tree는 regression tree와 마찬가지로 각 분할 영역에 대한 불순도를 측정해가며 모델링을 한다. 불순도(impurity)를 측정하는 대표적인 지표는 엔트로피(entropy), 지니계수(Gini Index)이다. CART의 경우 하나의 상위 노드당 두개의 하위 노드가 발생하므로 이진 분류 문제로 간주하여 불순도는 다음과 같이 계산할 수 있다.  $A$ 는 분할 영역이며,  $p$ 는 해당 파티션 내에서 오분류된 데이터의 비율이다.

$$I(A) = -p \log_2(p) - (1-p) \log_2(1-p)$$

완성된 tree에서 마지막 child node(leaf, 잎)의 예측값은 Y의 majority class를 선택한다. 즉, voting 방식을 이용하는 것이다.

날씨 데이터를 CART 알고리즘에 적용시켜 그날 야외활동을 할 수 있는지 아닌지 판단해보자.

### a) import dataset

```
# import package
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import tree

# import dataset
mydata = pd.read_csv("https://raw.githubusercontent.com/yohanesgultom/machine-learning-
assignment/master/playtennis.data")
mydata.head()
```

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

NOTE)

데이터 주소

<https://raw.githubusercontent.com/yohanesgultom/machine-learning-assignment/master/playtennis.data>

### b) data preprocessing

```
for i, name in enumerate(mydata.columns):
    print(f'{i+1} 번째 변수 : {name}')
    print(mydata[name].value_counts())
    print("\n")
```

1 번째 변수 : Outlook

Sunny 5

Rain 5

Overcast 4

Name: Outlook, dtype: int64

```
2 번째 변수 : Temperature
Mild      6
Hot       4
Cool      4
Name: Temperature, dtype: int64
```

```
3 번째 변수 : Humidity
High      7
Normal    7
Name: Humidity, dtype: int64
```

```
4 번째 변수 : Wind
Weak      8
Strong    6
Name: Wind, dtype: int64
```

```
5 번째 변수 : PlayTennis
Yes       9
No        5
Name: PlayTennis, dtype: int64
```

모든 변수가 범주형 변수이다. 따라서, 데이터 프레임으로 데이터를 변환 후 라벨 인코딩과 원-핫 인코딩을 한다.

```
# DataFrame
mydf = pd.DataFrame(data = mydata.values, columns = mydata.columns)

# One-Hot Encoding
mydf = pd.get_dummies(data = mydf, columns = ['Outlook', 'Temperature', 'Humidity', 'Wind'],
                      prefix = ['Outlook', 'Temperature', 'Humidity', 'Wind'])

# Label Encoding
encoder = LabelEncoder()
mydf['PlayTennis'] = encoder.fit_transform(mydf['PlayTennis'])

mydf.head(3)
```

	PlayTennis	Outlook_Overcast	Outlook_Rain	Outlook_Sunny	Temperature_Cool	Temperature_Hot	Temperature_Mild	Humidity_High	Humidity_Normal	Wind_Strong	Wind_Weak
0	0	0	0	1	0	1	0	1	0	0	1
1	0	0	0	1	0	1	0	1	0	1	0
2	1	1	0	0	0	1	0	1	0	0	1

NOTE)  
R에서 as.factor() 함수에 대해 python에선 인코딩 과정이다.

```
# X/y 분리
X = mydf.iloc[:,1: ].values
y = mydf.iloc[:, 0].values
```

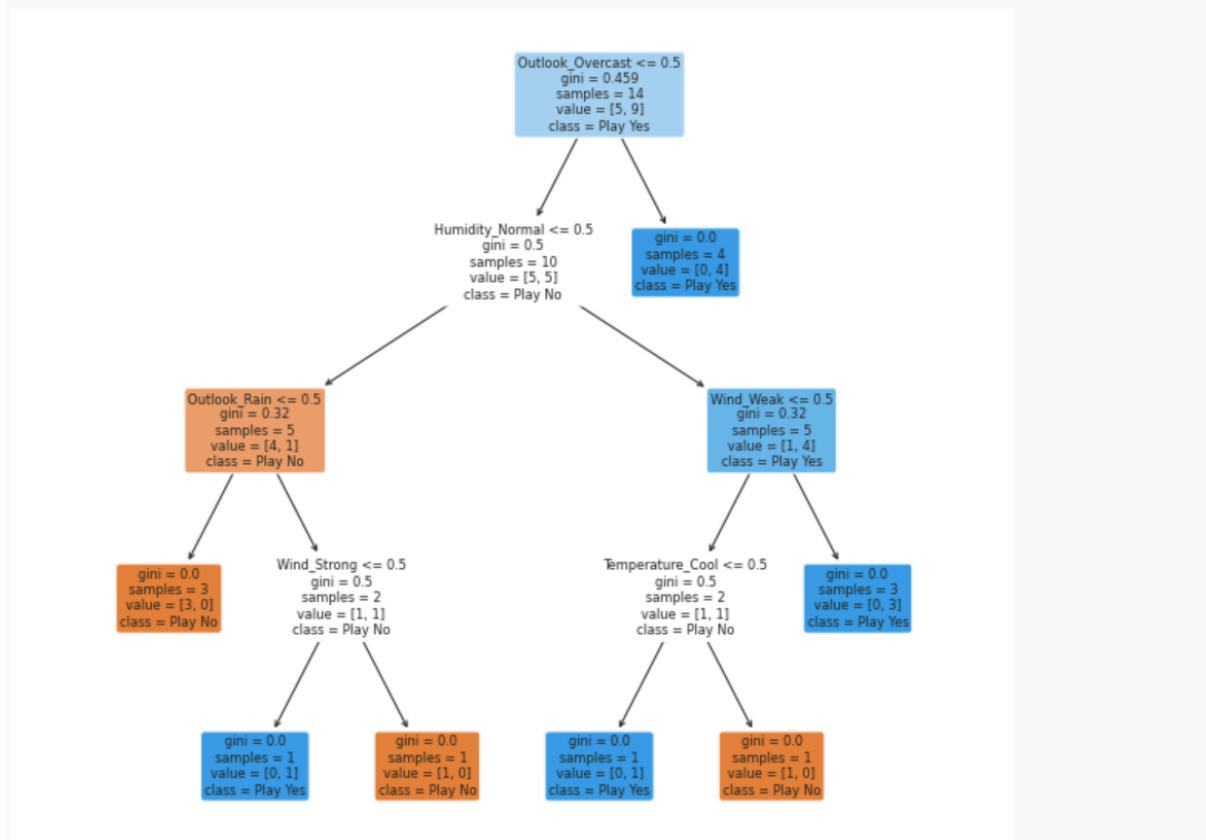
### c) model fitting

```
# CART Algorithm for classification
cart = DecisionTreeClassifier(criterion = 'gini', random_state = 0)
cart.fit(X, y)
```

### d) Test model

학습된 모델을 이용하여 트리를 plot으로 그려본다.

```
# Decision Tree visualization
fig = plt.figure(figsize = (10, 10))
_=tree.plot_tree(cart,
                  feature_names = mydf.columns[1:],
                  class_names = np.array(['Play No', 'Play Yes']),
                  impurity = True,
                  rounded = True,
                  filled = True
                 )
```



알고리즘의 성능을 판단하기 위해 confusion matrix와 오분류를 계산한다.

```
# confusion matrix
from sklearn.metrics import confusion_matrix
mtx = confusion_matrix(y, cart.predict(X))

label = ['Play No', 'Play Yes']
table = pd.DataFrame(mtx, columns = label, index = label)
table
```

	Play No	Play Yes
Play No	5	0
Play Yes	0	9

오분류은 0으로 모든 클래스를 정확하게 분류한다.

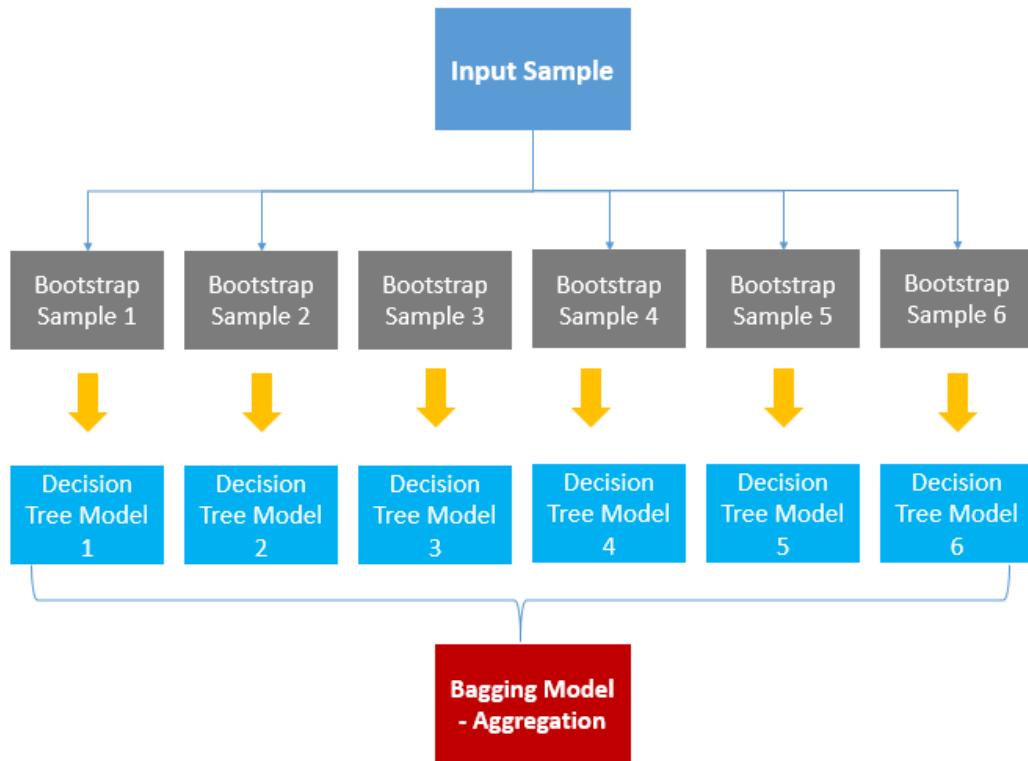
```
# 오분류율  
from sklearn.metrics import accuracy_score  
print(1-accuracy_score(y, cart.predict(X)))  
  
0.0
```

## 6. Ensemble

앙상블 방법이란, 머신러닝의 한 방법으로 문제해결을 위한 여러가지 모델들을 조합시켜 모델 훈련을 통해 최고의 성능을 보이는 조합을 찾아내는 것이다. 예를 들어, 분석의 상황에서 회귀 모델과 분류 등의 문제 사이에서 고민을 하게 되는데, 그 중 하나를 선택하는 것이 아닌 여러 모델의 조합을 통해 예측을 수집한다. 이 과정을 앙상블이라고 한다.

### 1) bagging

앙상블 모델을 학습하는 방법에는 각기 다른 훈련 알고리즘을 사용하는 방법과 같은 알고리즘을 사용하고 훈련 세트의 서브셋을 무작위로 구성하여 알고리즘에 각기 다르게 학습시키는 방법이 있다. 두번째 방법에서 훈련 세트의 중복을 허용하여 샘플링하는 방식을 배깅(bagging)이라고 한다.



배깅은 샘플을 여러번 뽑아(Bootstrap) 각 모델을 학습시켜 결과물을 집계(Aggregation)하는 방법으로 Bootstrap Aggregation의 약자이다. 데이터로부터 bootstrap(복원 랜덤 샘플링)을 한 뒤, 이 데이터로 모델을 학습한다. 그리고 학습된 모델의 결과를 집계하여 최종 결과 값을 구하는데, 분류의 경우 최빈값(mode)를 계산하고 회귀에 대해서는 평균(mean)을 계산한다. 개별적인 알고리즘은 훈련 세트로 훈련시킨 것보다 크게 편향되어 있지만 최빈값이나 평균을 계산함으로써 편향과 분산이 모두 감소한다. 일반적으로 앙상블의 결과는 원본 데이터셋으로 하나의 알고리즘을 훈련시킬 때와 비교해 편향은 비슷하지만 분산은 줄어든다.

sklearn은 배깅을 위해 간편한 API로 구성된 BaggingClassifier(회귀의 경우 BaggingRegressor)를 제공한다.

다음은 결정 트리(Decision Tree) 알고리즘 500개의 앙상블을 훈련시키는 코드이다. 각 알고리즘은 훈련세트에서 중복을 허용하여 무작위로 선택된 100개의 샘플로 훈련된다.

NOTE)

통계학에서는 중복을 허용한 리샘플링(resampling)을 부트스트래핑(bootstrapping)이라고 한다.

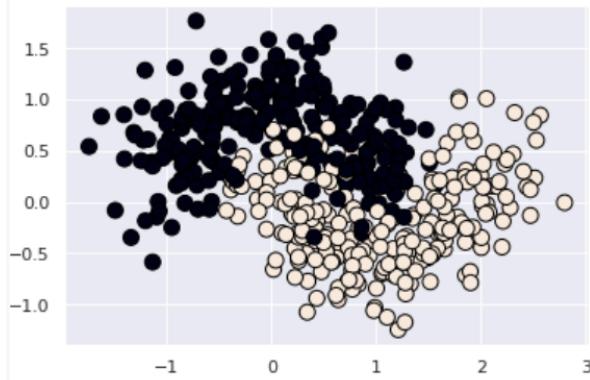
### a) import dataset

moons dataset을 이용하여 배깅을 학습하도록 한다. make\_moons() 함수를 사용하여 데이터셋을 만들 수 있다.

```
# import dataset
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)

# visualization dataset
plt.scatter(X[:,0], X[:,1], marker = 'o', c=y, s=100, edgecolor = "black")
plt.show()
```



NOTE)

moons 데이터셋은 사이킷런의 make\_moons 함수를 사용해서 만든 두 개의 반달 모양 데이터셋이다.

### b) data preprocessing

```
# split train, test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

### c) model fitting

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

mybagging = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
mybagging.fit(X_train, y_train)
y_pred = mybagging.predict(X_test)
```

### d) Test model

분류의 정확도를 계산하여 모델의 성능을 판단한다.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))

0.904
```

약 90%의 정확도를 보이는 알고리즘이다. 다음으로 단일 결정 트리의 결정 경계와 500개의 트리를 사용한 배깅 앙상블의 결정 경계와 성능을 비교해보도록 한다.

```
# single Decision Tree
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))

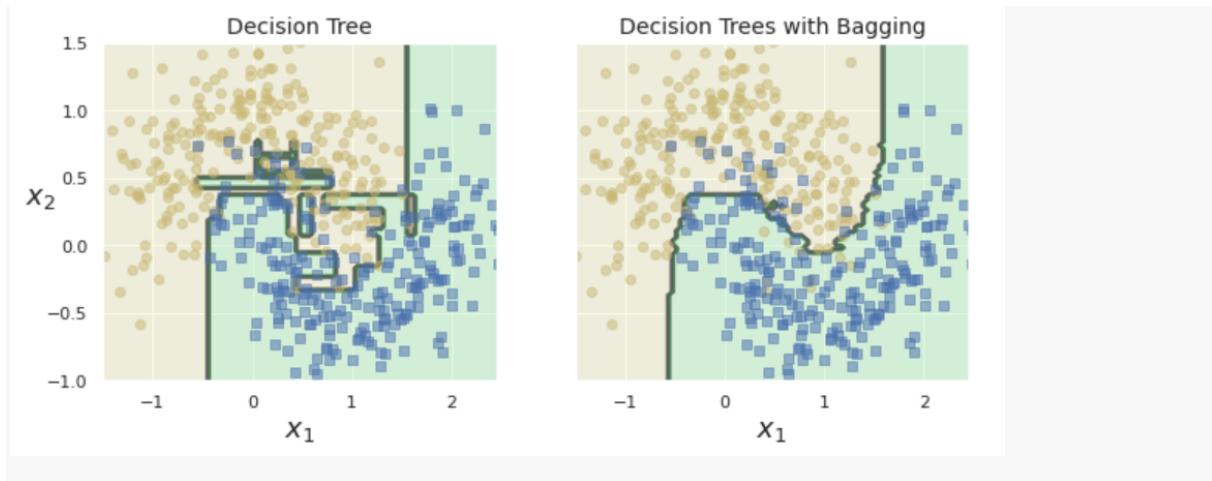
0.856
```

단일 결정 트리는 85.6%의 정확도를 보여주며 배깅 앙상블이 더 좋은 성능을 보여줄 수 있다. 마지막으로 시각화를 통해 결정 경계를 보도록 한다.

```
# decision boundary 를 그리기 위한 code
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58', '#4c4c7f', '#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)

fig, axes = plt.subplots(ncols=2, figsize=(10,4), sharey=True)
plt.sca(axes[0])
plot_decision_boundary(tree_clf, X, y)
plt.title("Decision Tree", fontsize=14)
plt.sca(axes[1])
plot_decision_boundary(mybagging, X, y)
plt.title("Decision Trees with Bagging", fontsize=14)
plt.ylabel("")
plt.show()
```



배깅 양상블의 예측이 단일 결정 트리의 예측보다 결정 경계가 덜 불규칙하며 분류 예측의 정확도가 높다.

## 2) boosting

boosting은 약한 학습기(weak learner, 랜덤 추출보다 조금 더 높은 성능을 내는 알고리즘)을 여러 개 연결하여 강한 학습기(strong learner, 높은 정확도를 내는 알고리즘)를 만드는 양상블 방법이다.

회귀 모델에서는 피팅이 더 개선될 수 있는지 알아보기 위해 잔차를 종종 사용한다. 부스팅은 이런 개념을 더 발전시켜, 이전 모델이 갖는 오차를 줄이는 방향으로 다음 모델을 연속적으로 생성한다. 에이다부스트(AdaBoost)와 그레디언트 부스팅(gradient boosting)은 가장 자주 사용되는 변형된 형태의 부스팅 알고리즘이다.

Terminology	
양상블(ensemble)	여러 모델들의 집합을 통해 예측 결과를 만들어 내는 것
부스팅(boosting)	연속된 라운드마다 잔차가 큰 레코드들에 가중치를 높여 일련의 모델들을 생성하는 일반적인 기법
에이다부스트(AdaBoost)	잔차에 따라 데이터의 가중치를 조절하는 부스팅의 초기 버전
그레디언트 부스팅(gradient boosting)	비용함수(loss function)를 최소화하는 방향으로 부스팅을 활용하는 좀 더 일반적인 형태
정규화(regularization)	비용함수에 모델의 파라미터 개수에 해당하는 벌점 항을 추가해 overfitting을 피하는 방법
하이퍼파라미터(hyperparameter)	알고리즘을 피팅하기 전에 미리 입력값을 결정해야 하는 파라미터

### a) Gradient Boosting

Gradient Boosting은 이전의 모델이 만든 잔여 오류(Residual Error)에 새로운 모델을 학습시키는 방법이다. 이 과정을 통해 오류를 줄여나갈 수 있다. 데이터를 생성하여 Decision Tree를 한 번만 학습하였을 때와 Gradient Boosting의 방법을 통해 학습시킨 결과를 비교해본다.

#### a-1) import dataset

```
# import package
import numpy as np
```

```
# 2 차 곡선 형태의 data 생성
np.random.seed(22)
X = 2*np.random.randn(100, 1)
y = 2*X**2 + X*4 + 10 + np.random.randn(100, 1)
```

### a-2) model fitting

DecisionTreeRegressor에 data를 train 시킨다.

```
from sklearn.tree import DecisionTreeRegressor
mytree = DecisionTreeRegressor(max_depth = 2)
mytree.fit(X, y)

DecisionTreeRegressor(max_depth=2)
```

첫번째 모델에서 생긴 잔여 오차에 두번째 DecisionTreeRegressor를 훈련시킨다.

```
y2 = y - mytree.predict(X)[:, None] # 2 차원의 array 형태로 만들
mytree2 = DecisionTreeRegressor(max_depth = 2)
mytree2.fit(X, y2)

DecisionTreeRegressor(max_depth=2)
```

두번째 모델이 만든 잔여 오차에 세번째 회귀 모델을 훈련시킨다.

```
y3 = y2 - mytree2.predict(X)[:, None]
mytree3 = DecisionTreeRegressor(max_depth = 2)
mytree3.fit(X, y3)

DecisionTreeRegressor(max_depth=2)
```

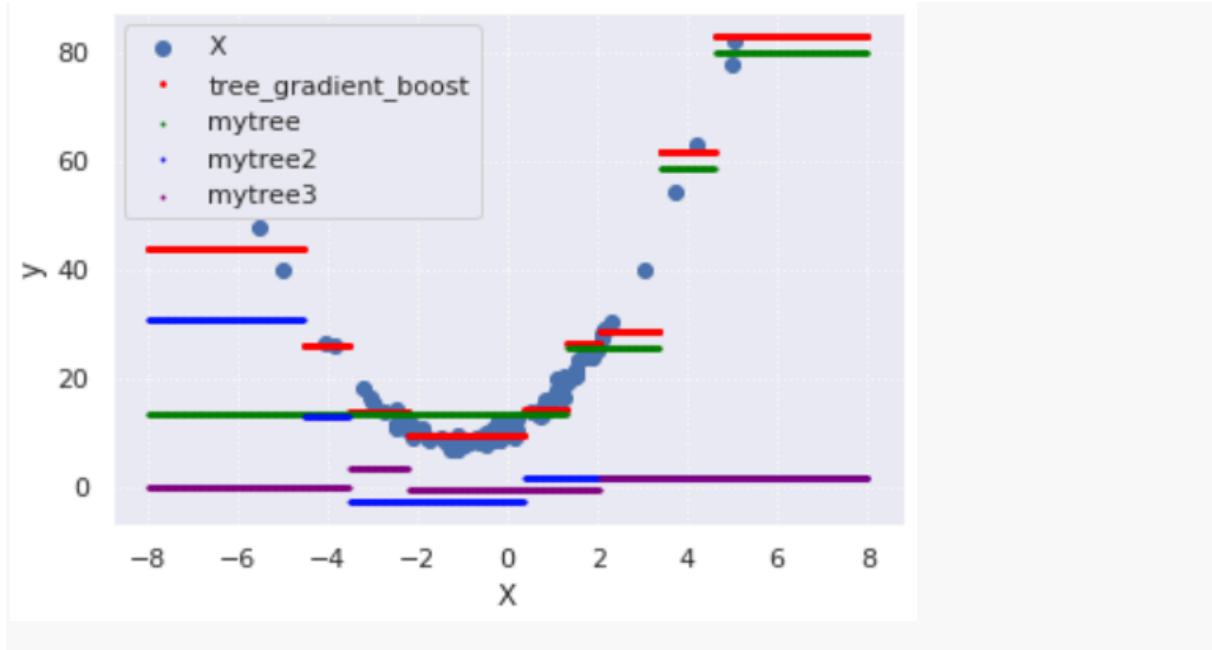
이제 세 개의 트리를 포함하는 양상을 모델이 생겼다. 새로운 샘플에 대한 예측을 만들기위해 모든 트리의 예측값을 더한다.

```
X_new = np.linspace(-8, 8, 1000).reshape(-1, 1)
y_pred = sum(tree.predict(X_new) for tree in (mytree, mytree2, mytree3))
```

### a-3) Test model

세 트리의 예측과 양상을의 예측을 시각화하여 비교해보자.

```
plt.scatter(X, y)
plt.scatter(X_new, y_pred, c = 'red', s=3)
plt.scatter(X_new, mytree.predict(X_new), c='green', s=1)
plt.scatter(X_new, mytree2.predict(X_new), c='blue', s=1)
plt.scatter(X_new, mytree3.predict(X_new), c='purple', s=1)
plt.xlabel("X")
plt.ylabel("y")
plt.grid(linestyle = "dotted")
plt.legend(['X', 'tree_gradient_boost', 'mytree', 'mytree2', 'mytree3'], loc = 'upper left')
plt.show()
```



양상블 모델의 예측값이 트리를 이용한 예측값보다 좋아지고 있다는 것을 알 수 있다. 훈련 횟수를 좀 더 추가하여 성능이 좋아지는지 확인하도록 한다.

```
# import package
import numpy as np

# data
X = 2*np.random.randn(100, 1)
y = 2*X**2 + X*4 + 10 + np.random.randn(100, 1)

treelist = []

# initial value
mytree = DecisionTreeRegressor(max_depth = 2)
treelist.append(mytree)
mytree.fit(X, y)

# Gradient Boosting 을 10 번 실행하는 반복문
for i in range(0, 10):

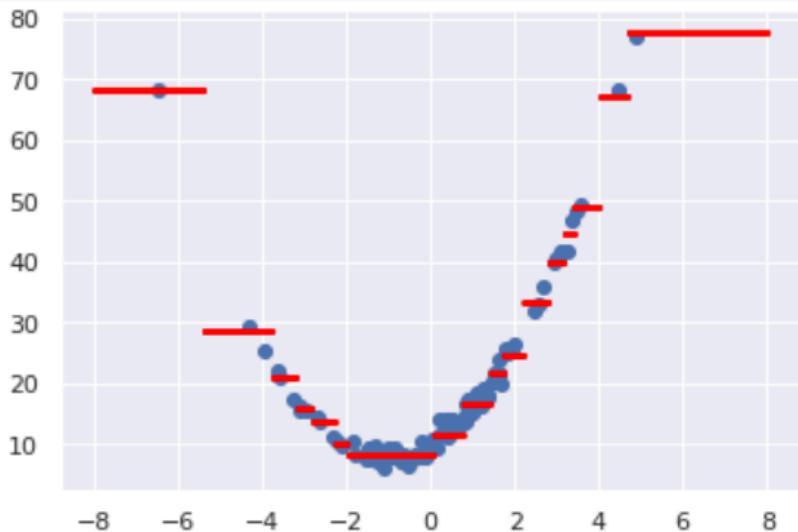
    # 잔여 오차 계산
    y2 = y - mytree.predict(X)[:, None]
    mytree = DecisionTreeRegressor(max_depth = 2)
    treelist.append(mytree)
    mytree.fit(X, y2)

    # 다음 iteration 을 위해 y update
    y = y2

# 반복문에 의해 y 가 달라졌으므로 처음의 y 를 다시 지정
y = 2*X**2 + X*4 + 10 + np.random.randn(100, 1)

X_new = np.linspace(-8, 8, 1000).reshape(-1, 1)
y_pred = sum(tree.predict(X_new) for tree in treelist)
```

```
# visualization
plt.scatter(X, y)
plt.scatter(X_new, y_pred, c = 'red', s=3)
plt.show()
```



10번 반복 후 예측력이 훨씬 좋아졌다. 다음은 sklearn에서 제공하는 Gradient Boosting package를 이용한 코드이다.

```
import warnings
warnings.filterwarnings(action='ignore')

# 2 차 곡성 형태의 data 생성
np.random.seed(22)
X = 2*np.random.randn(100, 1)
y = 2*X**2 + X*4 + 10 + np.random.randn(100, 1)

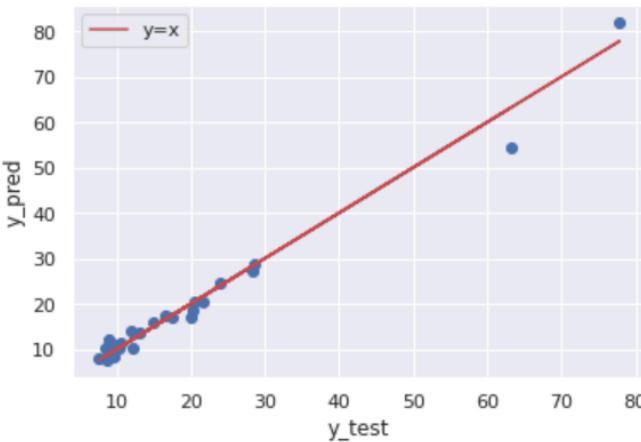
# test/train split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

# model fitting
from sklearn.ensemble import GradientBoostingRegressor

mygbt = GradientBoostingRegressor(learning_rate = 0.1, random_state = 0)
mygbt.fit(X_train, y_train)
y_pred = mygbt.predict(X_test)

# test model
plt.scatter(y_test, y_pred)
plt.plot(y_test, y_test, label = "y=x", color = "r")
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.legend()
plt.show()
```



### b) XGBoost

XGBoost는 부스팅에 관련된 여러 알고리즘에서 현재 가장 인기가 많다. 매우 빠른 속도와 정확도 등으로 머신러닝 경연 대회에서 자주 사용되는 도구 중의 하나이다.

#### b-1) import dataset

```
# import package
import pandas as pd
import numpy as np
import xgboost           # xgboost package가 install 되었는지 확인
import warnings
warnings.filterwarnings("ignore")

# data (n = 1000)
np.random.seed(22)
X = 2*np.random.randn(1000, 1)
y = 2*X**2 + X*4 + 10 + np.random.randn(1000, 1)
```

#### b-2) data preprocessing

```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

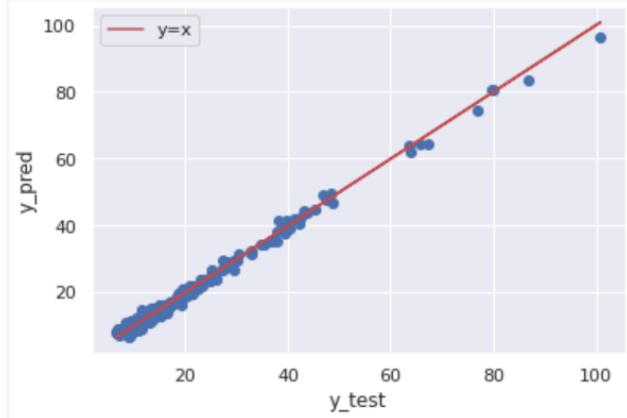
#### b-3) model fitting

```
myxgb = xgboost.XGBRegressor(random_state=0)
myxgb.fit(X_train, y_train)
y_pred = myxgb.predict(X_test)
```

#### b-4) Test model

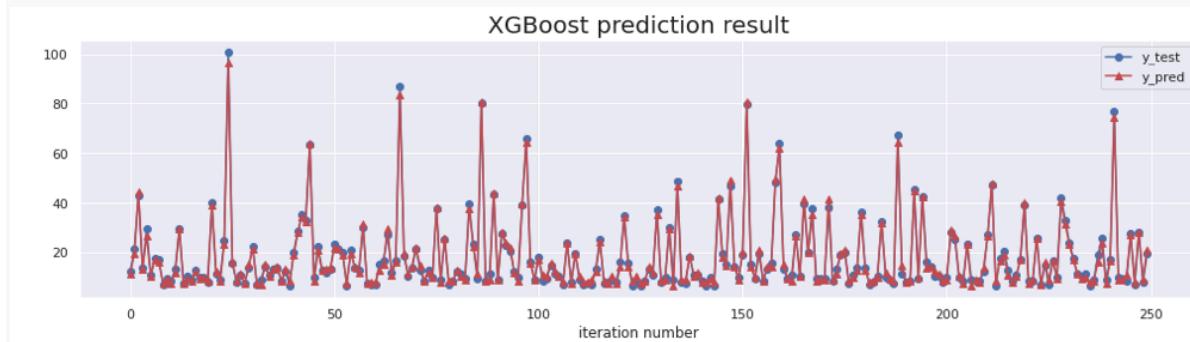
```
# visualization 1
plt.scatter(y_test, y_pred)
plt.plot(y_test, y_test, label = "y=x", color = "r")
plt.xlabel("y_test")
plt.ylabel("y_pred")
```

```
plt.legend()
plt.show()
```



예측값이 실제값을 잘 추정하고 있음을 확인할 수 있다.

```
# visualization 2
plt.figure(figsize = (17, 4))
plt.plot(y_test, marker = "o", color = "b", label = "y_test")
plt.plot(y_pred, marker = "^", color = "r", label = "y_pred")
plt.title("XGBoost prediction result", size = 20)
plt.xlabel("iteration number")
plt.legend()
plt.show()
```



### cf. hyperparameter

learning rate는 XGBoost의 hyperparameter(tuning parameter) 중 하나이다. 즉, learning rate를 변화시키면서 최적의 모델을 찾기 위한 과정이 필요하다. 일반적으로 learning rate의 값은 0.1, 0.05, 0.01, 0.001을 이용한다.

```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.",
                     data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

```

# X, y 생성
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

from sklearn.ensemble import GradientBoostingRegressor

# Learning rate = 0.1
gbdt1 = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.1,
random_state=42)
gbdt1.fit(X, y)

# Learning rate = 0.05
gbdt2 = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.05,
random_state=42)
gbdt2.fit(X, y)

# Learning rate = 0.01
gbdt3 = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.01,
random_state=42)
gbdt3.fit(X, y)

# Learning rate = 0.001
gbdt4 = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.001,
random_state=42)
gbdt4.fit(X, y)

# visualization
plt.figure(figsize=(11,11))

plt.subplot(221)
plot_predictions([gbdt1], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("learning_rate={}, n_estimators={}".format(gbdt1.learning_rate, gbdt1.n_estimators),
fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)

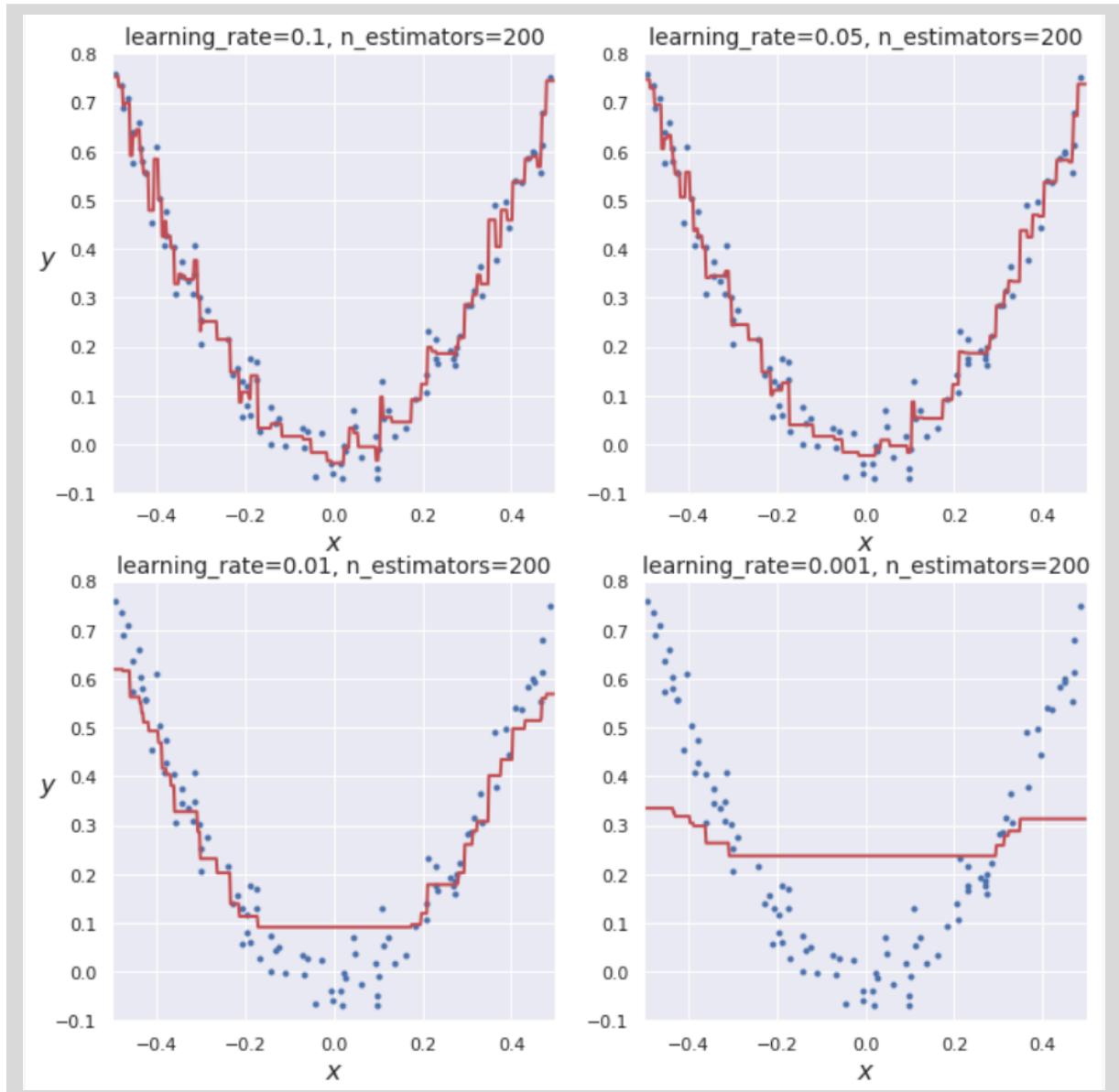
plt.subplot(222)
plot_predictions([gbdt2], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("learning_rate={}, n_estimators={}".format(gbdt2.learning_rate, gbdt2.n_estimators),
fontsize=14)
plt.xlabel("$x$", fontsize=16)

plt.subplot(223)
plot_predictions([gbdt3], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("learning_rate={}, n_estimators={}".format(gbdt3.learning_rate, gbdt3.n_estimators),
fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)

plt.subplot(224)
plot_predictions([gbdt4], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("learning_rate={}, n_estimators={}".format(gbdt4.learning_rate, gbdt4.n_estimators),
fontsize=14)
plt.xlabel("$x$", fontsize=16)

plt.show()

```



jmlkin

### cf. Gradient Boosting VS XGBoost

Gradient Boosting과 XGBoost를 비교해보자.

```
# RMSE
from sklearn.metrics import mean_squared_error

# Gradient Boosting
from sklearn.ensemble import GradientBoostingRegressor
gbt = GradientBoostingRegressor(max_depth=2, n_estimators=120, random_state=42)
gbt.fit(X_train, y_train)
y_pred = gbt.predict(X_test)
%timeit GradientBoostingRegressor().fit(X_train, y_train)
print("validation error", mean_squared_error(y_test, y_pred))

36.3 ms ± 292 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
validation error 1.1802893464703095

# XGBoost
xgb = xgboost.XGBRegressor()
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
%timeit xgboost.XGBRegressor().fit(X_train, y_train)
print("validation error", mean_squared_error(y_test, y_pred))

79.2 ms ± 11.5 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
validation error 1.44169323738124
```

위의 데이터에 대해서는 계산 속도와 예측의 정확성을 고려할 때, Gradient Boosting이 더 좋은 성능을 보이고 있다.

### 3) random forest

Bagging은 복원 추출 방식으로 데이터를 만들기 때문에 중복되는 데이터가 발생한다. 따라서, 독립성이 깨지는 문제가 발생하는데, 독립성을 만족시키지 못할 우 모델간의 공분산이 발생하여 비슷한 tree가 만들어질 가능성이 높아진다.

그렇다면, 어떻게 분산을 줄일까?

random forest는 Bagging처럼 데이터를 반복 복원추출하며, 변수 또한 random하게 추출하여 다양한 모델을 만든다. 변수를 random하게 추출하여 모델을 다양하게 만드는 과정을 통해 모델간 공분산을 줄이는 효과가 나오는 것이다.

kaggle에서 제공하는 데이터를 이용하여 random forest를 실습한다.

#### a) import dataset

```
# import package
import pandas as pd

# import dataset
mydata = pd.read_csv("./otto_data.csv") # 실행중인 주피터 노트북과 동일한 위치에 csv 파일을 저장한 후
불러오는 방법
mydata.head()
```

	id	feat_1	feat_2	feat_3	feat_4	feat_5	feat_6	feat_7	feat_8	feat_9	...	feat_85	feat_86	feat_87	feat_88	feat_89	feat_90	feat_91	feat_92	feat_93	target
0	1	1	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	Class_1
1	2	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	Class_1
2	3	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	Class_1
3	4	1	0	0	1	6	1	5	0	0	...	0	1	2	0	0	0	0	0	0	Class_1
4	5	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	1	0	0	0	Class_1

5 rows × 95 columns

id는 고유 아이디이며 필요한 feature가 아니므로 삭제한다. feat\_1 ~ feat\_93 은 feature이며 target columns은 클래스가 9개(Class 1 ~ Class 9)로 나누어져 있는 target이다.

#### b) data preprocessing

```
# 불필요한 변수 제거
mydata = mydata.drop(['id'], axis = 1)

target을 수치형 변수로 변환하기 위해 라벨인코딩을 한다.
```

```
# Label Encoding
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(mydata["target"])
mydata["target"] = encoder.transform(mydata["target"])

print("라벨링 결과")
for i, label in enumerate(encoder.classes_):
    print(label, '→', i)
```

라벨링 결과

```
Class_1 -> 0
Class_2 -> 1
Class_3 -> 2
Class_4 -> 3
Class_5 -> 4
Class_6 -> 5
Class_7 -> 6
Class_8 -> 7
Class_9 -> 8
```

다음으로 학습위해 데이터를 train/test로 분리한다.

```
# feature/target 분리
feature_columns = list(mydata.columns.difference(['target']))
X = mydata[feature_columns]
y = mydata['target']

# train/test 분리
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

(49502, 93) (12376, 93) (49502,) (12376,)
```

### c) model fitting

```
# Random forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

myrf = RandomForestClassifier(n_estimators = 20, max_depth = 5, random_state = 42)
myrf.fit(X_train, y_train)

RandomForestClassifier(max_depth=5, n_estimators=20, random_state=42)
```

### d) Test model

```
y_pred = myrf.predict(X_test)
print(accuracy_score(y_test, y_pred))

0.6010019392372333
```

정확도가 약 60% 나왔다. 성능을 높이기 위해 sample을 늘려본다.

```
# sample = 100
myrf = RandomForestClassifier(n_estimators = 100, max_depth = 5, random_state = 42)
myrf.fit(X_train, y_train)
y_pred = myrf.predict(X_test)
print(accuracy_score(y_test, y_pred))

0.6240303813833226
```

```
# sample = 200
myrf = RandomForestClassifier(n_estimators = 200, max_depth = 5, random_state = 42)
myrf.fit(X_train, y_train)
y_pred = myrf.predict(X_test)
print(accuracy_score(y_test, y_pred))

0.6157886231415644

# sample = 400
myrf = RandomForestClassifier(n_estimators = 400, max_depth = 5, random_state = 42)
myrf.fit(X_train, y_train)
y_pred = myrf.predict(X_test)
print(accuracy_score(y_test, y_pred))

0.616596638655462

# sample = 500
myrf = RandomForestClassifier(n_estimators = 500, max_depth = 5, random_state = 42)
myrf.fit(X_train, y_train)
y_pred = myrf.predict(X_test)
print(accuracy_score(y_test, y_pred))

0.6168390433096316
```

sample의 갯수가 늘어날수록 정확도가 비례하여 좋아지지는 않는다는 것을 확인했다. 일반적으로 RF에서 max\_depth = 5로 지정한다.

### cf. Grid Search CV

learning rate 이외에 여러 Boosting 모델에선 다양한 hyperparameter가 존재한다. Hyperparameter는 모델을 세부적으로 tuning하여 최적의 성능을 내는 모델을 찾는 과정이다. 가장 좋은 성능을 내는 hyperparameter 조합을 찾는 것은 매우 많은 경우의 수가 존재하기 때문에 Grid Search를 사용하는 것이 좋다. 설정하고자 하는 hyperparameter와 그 값을 지정하면 된다. 그 다음 가능한 모든 hyperparameter 조합에 대해 Cross validation(교차검증)을 사용해 평가하게 된다. 예를 들어 다음 코드는 RandomForestRegressor에 대한 최적의 hyperparameter 조합을 탐색한다.

```
import warnings
from sklearn.datasets import load_boston, load_breast_cancer
from sklearn.model_selection import train_test_split
warnings.filterwarnings("ignore")

boston = load_boston()
X = boston.data
y = boston.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

param_grid = [
    {'n_estimators' : [100, 200, 300], 'max_features' : [2, 3, 4, 5]},
    {'bootstrap' : [False], 'n_estimators' : [100, 200, 300], 'max_features' : [4, 5, 6, 7, 8]}
]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv = 5,
                           scoring = 'neg_mean_squared_error',
                           return_train_score = True)

grid_search.fit(X_train, y_train)

GridSearchCV(cv=5, estimator=RandomForestRegressor(),
            param_grid=[{'max_features': [2, 3, 4, 5],
                         'n_estimators': [100, 200, 300]},
                        {'bootstrap': [False], 'max_features': [4, 5, 6, 7, 8],
                         'n_estimators': [100, 200, 300]}],
            return_train_score=True, scoring='neg_mean_squared_error')
```

NOTE)

grid search는 scoring에 지정된 함수의 값이 클수록 좋은 함수로 인식한다. 따라서, scoring = 'neg\_mean\_squared\_error'을 사용해야 가장 작은 RMSE 값이 제일 큰 값으로 인식되고, 나중에 다시 (-)를 곱하면 원래의 RMSE 값이 출력된다.

Param\_grid 설정에 따라 sklearn이 먼저 첫 번째 dict에 있는 n\_estimators와 max\_features 조합인 3x4=12개를 평가한 다음, 두 번째 dict에 있는 조합인 3x5=15개를 시도한다. 다른 조합을 시도해보고 싶다면 dict을 추가해주면 된다. 첫 번째 조합과 두 번째 조합을 모두 탐색하면 hyperparameter의 값이 12+15=27개의 모델을 훈련하고 각각 다섯번씩 모델을 훈련한다. (5-cross validation을 사용하기 때문이다.) 따라서, 총 27x5=135 번 훈련하게 된다. 훈련 후 다음과 같이 최적의 조합을 얻을 수 있다.

```
grid_search.best_params_
{'max_features': 5, 'n_estimators': 200}
```

이 예시에서는 max\_features가 6, n\_estimators가 30일 때 최적의 모델을 훈련한다. (코드를 실행할 때마다 다른 조합의 결과가 나올 것이다.)

```
grid_search.best_estimator_
RandomForestRegressor(max_features=5, n_estimators=200)
```

각 모델에 대한 테스트 결과도 확인할 수 있다.

```
cvres = grid_search.cv_results_
import numpy as np

for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)

3.887255654462174 {'max_features': 2, 'n_estimators': 100}
3.776059194183585 {'max_features': 2, 'n_estimators': 200}
3.7881747406631456 {'max_features': 2, 'n_estimators': 300}
3.717899076854966 {'max_features': 3, 'n_estimators': 100}
3.6295481020156855 {'max_features': 3, 'n_estimators': 200}
3.675060826318378 {'max_features': 3, 'n_estimators': 300}
3.6693029922647713 {'max_features': 4, 'n_estimators': 100}
3.598341097894671 {'max_features': 4, 'n_estimators': 200}
3.7147113205797693 {'max_features': 4, 'n_estimators': 300}
3.656112653362047 {'max_features': 5, 'n_estimators': 100}
3.573146109041531 {'max_features': 5, 'n_estimators': 200}
3.636625897504709 {'max_features': 5, 'n_estimators': 300}
3.6035155157802667 {'bootstrap': False, 'max_features': 4, 'n_estimators': 100}
3.6132262175157113 {'bootstrap': False, 'max_features': 4, 'n_estimators': 200}
3.6177687947774815 {'bootstrap': False, 'max_features': 4, 'n_estimators': 300}
3.6541491303899485 {'bootstrap': False, 'max_features': 5, 'n_estimators': 100}
3.6082750069862537 {'bootstrap': False, 'max_features': 5, 'n_estimators': 200}
3.587399202833019 {'bootstrap': False, 'max_features': 5, 'n_estimators': 300}
3.6979531338713096 {'bootstrap': False, 'max_features': 6, 'n_estimators': 100}
3.659903259913983 {'bootstrap': False, 'max_features': 6, 'n_estimators': 200}
3.6758632997981806 {'bootstrap': False, 'max_features': 6, 'n_estimators': 300}
3.8604627687039508 {'bootstrap': False, 'max_features': 7, 'n_estimators': 100}
3.803824163098649 {'bootstrap': False, 'max_features': 7, 'n_estimators': 200}
3.7994937206909514 {'bootstrap': False, 'max_features': 7, 'n_estimators': 300}
3.90873144922499 {'bootstrap': False, 'max_features': 8, 'n_estimators': 100}
3.8937018538469568 {'bootstrap': False, 'max_features': 8, 'n_estimators': 200}
3.905073590178799 {'bootstrap': False, 'max_features': 8, 'n_estimators': 300}
```

여러 boosting 모델을 정의하여 각 모델별 grid search를 진행해보도록 한다.

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost
from xgboost import XGBRegressor

def clf3():
    rf = RandomForestRegressor()
    gb = GradientBoostingRegressor()
    xg = XGBRegressor()

    return rf, gb, xg

from sklearn import model_selection

rf, gb, xg = clf3()

rf_parameters ={'max_depth' : [3,4,5,6,7,8] , 'n_estimators': [100, 200, 300],
                 'random_state':[99]}
gb_parameters ={'max_depth' : [4,5,7] , 'n_estimators': [100, 200, 300],
                 'learning_rate':[0.01, 0.1], 'random_state':[99]}
xg_parameters ={'max_depth' : [3,4,5,6] , 'n_estimators': [100, 200, 300],
                 'learning_rate':[0.01, 0.1], 'gamma': [0.5, 1, 2], 'random_state':[99]}

#RF에 대한 grid search, scoring 기준은 recall 으로, cross-validation 은 10 회 반복
grid_search_rf = model_selection.GridSearchCV ( estimator = rf, param_grid = rf_parameters,
scoring = 'recall', cv = 10 )

grid_search_rf.fit( X_train, y_train )

#best hyper parameter 값을 봄
best_rf = grid_search_rf.best_estimator_
best_rf

RandomForestRegressor(max_depth=3, random_state=99)

print(round(best_rf.score(X_test,y_test), 3))

0.823

from sklearn.metrics import r2_score
y_pred_rf = best_rf.predict(X_test)
r2_score(y_pred_rf, y_test)

0.7334430608741815

# Gradient Boosting
grid_search_gb = model_selection.GridSearchCV ( estimator = gb, param_grid = gb_parameters,
scoring = 'recall', cv = 10 )

grid_search_gb.fit( X_train, y_train )

#best hyper parameter 값을 봄
best_gb = grid_search_gb.best_estimator_
best_gb

GradientBoostingRegressor(learning_rate=0.01, max_depth=4, random_state=99)

```

```

print(round(best_gb.score(X_test,y_test), 3))

0.742

from sklearn.metrics import r2_score
y_pred_gb = best_gb.predict(X_test)
r2_score(y_pred_gb, y_test)

0.19733554245189855

# XGBoost
grid_search_xg = model_selection.GridSearchCV ( estimator = xg, param_grid = gb_parameters,
scoring = 'recall', cv = 10 )

grid_search_xg.fit( X_train, y_train )

#best hyper parameter 값을 받음
best_xg = grid_search_xg.best_estimator_
best_xg

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=99, ...)

print(round(best_xg.score(X_test,y_test), 3))

-0.165

from sklearn.metrics import r2_score
y_pred_xg = best_xg.predict(X_test)
r2_score(y_pred_xg, y_test)

-3.288927441342995

```



## 7. Study Case for classification

앞에서 배운 classification algorithm에 UCI에서 제공하는 Bank Marketing Dataset을 이용하여 'deposit'을 이용할 것인지 아닌지를 예측하는 모델을 만들고 각 모델별로 예측력을 비교해보자.

데이터는 Portuguese banking institution의 고객과 마켓팅에 대한 정보를 나타낸다. feature들은 고객의 정보와 전화를 이용한 deposit 광고, 고객과 사회의 당시 경제 정보이며, target은 deposit의 이용을 수락하였는지 아닌지에 대한 binary 변수이다.

feature	
1. age	(numeric)
2. job	type of job (categorical)
3. marital	marital status (categorical)
4. education	(categorical)
5. default	has credit in default? (categorical)
6. housing	has housing loan? (categorical)
7. loan	has personal loan? (categorical)
8. contact	contact communication type (categorical)
9. month	last contact month of year (categorical)
10. day_of_week	last contact day of the week (categorical)
11. duration	last contact duration, in seconds (numeric)
12. campaign	number of contacts performed during this campaign and for this client (numeric)
13. pdays	number of days that passed by after the client was last contacted from a previous campaign (numeric)
14. previous	number of contacts performed before this campaign and for this client (numeric)
15. poutcome	outcome of the previous marketing campaign (categorical)
16. emp.var.rate	employment variation rate (numeric)
17. cons.price.idx	consumer price index (numeric)
18. cons.conf.idx	consumer confidence index (numeric)
19. euribor3m	euribor 3 month rate (numeric)
20. nr.employed	number of employees (numeric)

NOTE)

Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

target	
21. y	has the client subscribed a term deposit? (binary)

### a) import dataset

```
import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import pandas as pd
bank = pd.read_csv("./bank-additional-full.csv", sep = ";")
bank.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4

데이터의 갯수는 41188개이며 feature의 개수는 20개, target은 1개이다. 다음으로, feature와 target에 대한 변수 타입을 관찰한다.

```
bank.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   age               41188 non-null   int64  
 1   job               41188 non-null   object 
 2   marital           41188 non-null   object 
 3   education         41188 non-null   object 
 4   default           41188 non-null   object 
 5   housing           41188 non-null   object 
 6   loan              41188 non-null   object 
 7   contact           41188 non-null   object 
 8   month             41188 non-null   object 
 9   day_of_week       41188 non-null   object 
 10  duration          41188 non-null   int64  
 11  campaign          41188 non-null   int64  
 12  pdays             41188 non-null   int64  
 13  previous          41188 non-null   int64  
 14  poutcome          41188 non-null   object 
 15  emp.var.rate      41188 non-null   float64
 16  cons.price.idx    41188 non-null   float64
 17  cons.conf.idx     41188 non-null   float64
 18  euribor3m         41188 non-null   float64
 19  nr.employed       41188 non-null   float64
 20  y                 41188 non-null   object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

모델링을 하기 이전 단계에서 type이 "object"인 feature는 라벨링과정이 필요하다. duration은 y가 관측된 이후 작성된 데이터이므로 duration에 의해 모델의 성능이 영향을 받는다. 따라서, 해당 변수는 제거 후 모델링을 하도록 한다.

```
# duration 제거
bank = bank.drop(['duration'], axis = 1)
```

또한, 몇가지 범주형 변수에 대하여 histogram 을 그려 각 feature 에 대해 살펴보도록 한다.

```
# Histogram
plt.figure(figsize = (16, 15))
plt.subplots_adjust(hspace = 1)

# job
plt.subplot(321)
bank['job'].value_counts(sort = False).plot.bar(rot = 45)
plt.title('Histogram of job')
plt.xlabel('job')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)

# marital
plt.subplot(322)
bank['marital'].value_counts(sort = False).plot.bar(rot = 45)
plt.title('Histogram of marital')
plt.xlabel('marital')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)

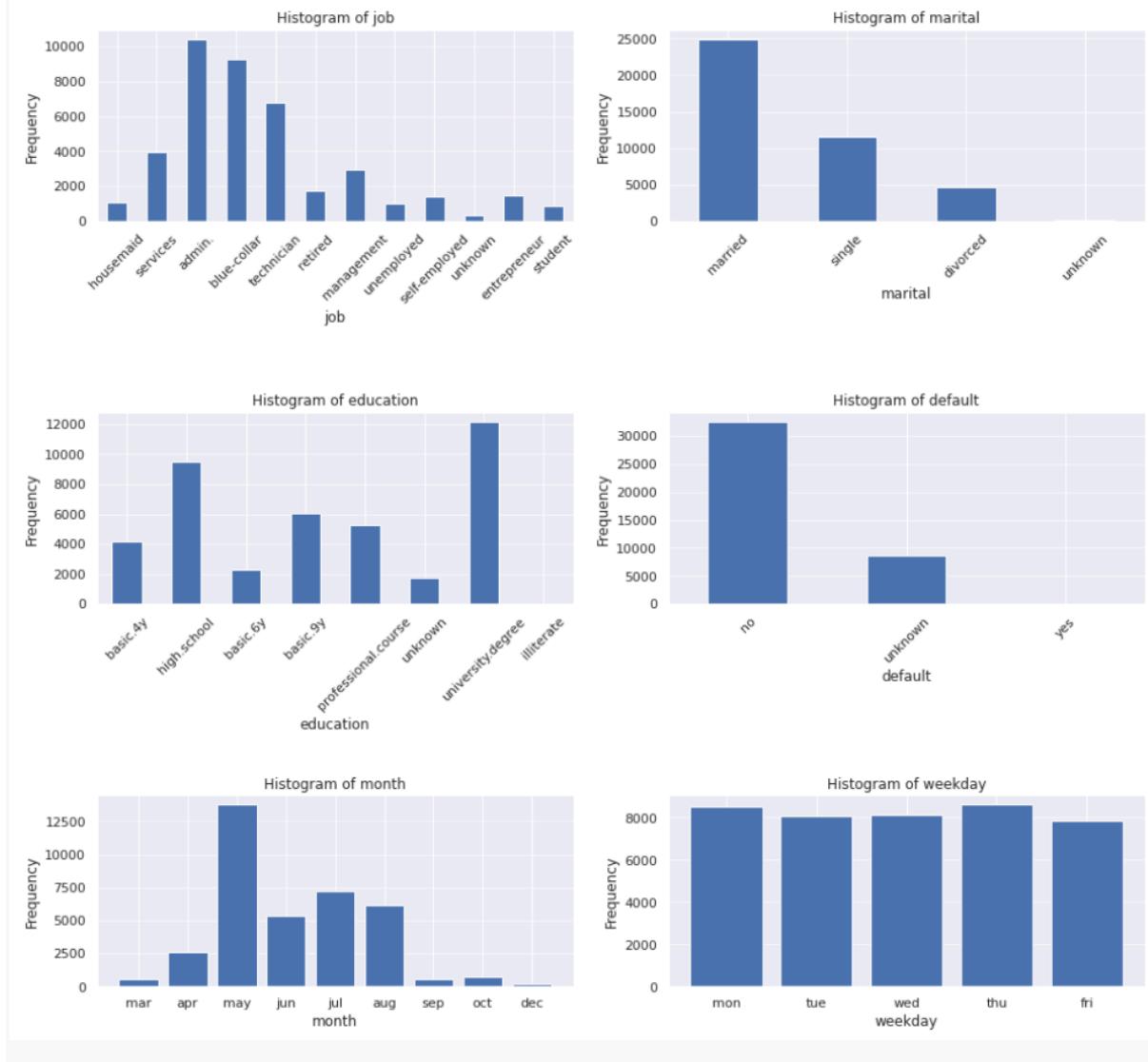
# education
plt.subplot(323)
bank['education'].value_counts(sort = False).plot.bar(rot = 45)
plt.title('Histogram of education')
plt.xlabel('education')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)

# default
plt.subplot(324)
bank['default'].value_counts(sort = False).plot.bar(rot = 45)
plt.title('Histogram of default')
plt.xlabel('default')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)

# month
plt.subplot(325)
# changing the order of x-axis label using loc
month_order = ['mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'dec']
df_month = bank['month'].value_counts()
df_month = df_month.loc[month_order]
plt.bar(df_month.index, df_month.values)
plt.title('Histogram of month')
plt.xlabel('month')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)

# day_of_week
plt.subplot(326)
# changing the order of x-axis label using loc
weekday_order = ['mon', 'tue', 'wed', 'thu', 'fri']
df_weekday = bank['day_of_week'].value_counts()
df_weekday = df_weekday.loc[weekday_order]
plt.bar(df_weekday.index, df_weekday.values)
```

```
plt.title('Histogram of weekday')
plt.xlabel('weekday')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)
```

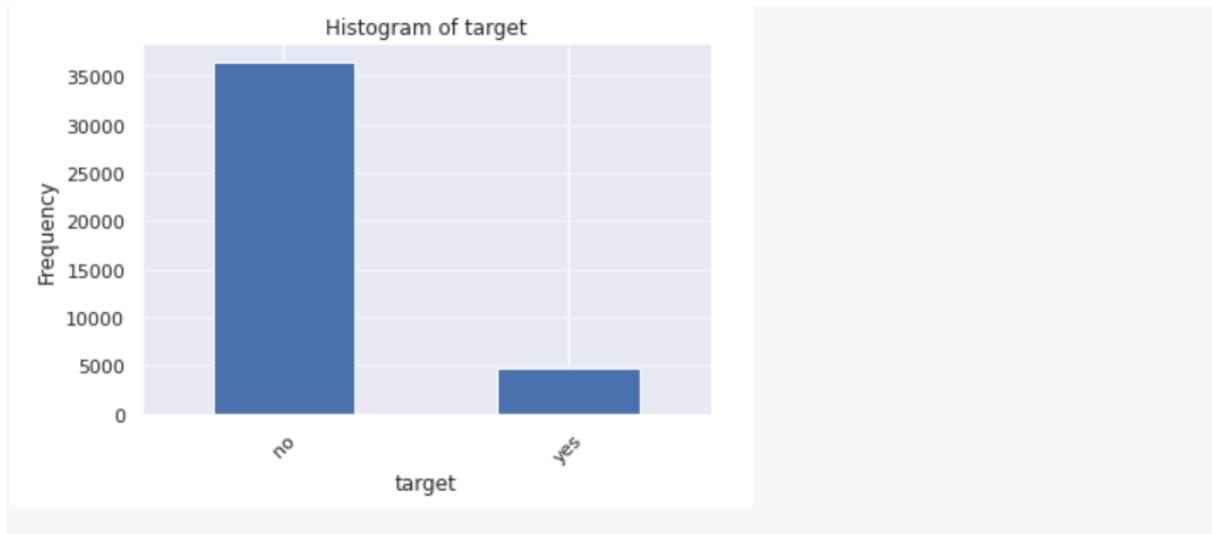


다음으로 target에 대한 그래프를 그린다.

```
bank['y'].value_counts()
```

```
no      36548
yes     4640
Name: y, dtype: int64
```

```
# target
bank['y'].value_counts(sort = False).plot.bar(rot = 45)
plt.title('Histogram of target')
plt.xlabel('target')
plt.ylabel('Frequency')
plt.grid(axis = 'y', alpha = 0.7)
```



'no'의 빈도수가 'yes'에 비해 많다.

### b) data preprocessing

데이터에 결측치가 없는지 확인한다.

```
bank.isnull().sum()

age          0
job          0
marital      0
education    0
default      0
housing      0
loan          0
contact      0
month         0
day_of_week   0
campaign     0
pdays        0
previous     0
poutcome     0
emp.var.rate 0
cons.price.idx 0
cons.conf.idx 0
euribor3m    0
nr.employed  0
y             0
dtype: int64
```

결측치가 존재하지 않는다. 범주형 변수들을 원-핫 인코딩 후 데이터를 분리한다.

```
# feature/target split
X = bank.drop('y', axis = 1)
Y = bank['y']

# DataFrame
mydf = pd.DataFrame(data = X.values, columns = X.columns)
```

```
# One-Hot Encoding
X = pd.get_dummies(data = mydf,
                    columns = ['job', 'marital', 'education', 'default', 'housing', 'loan',
                               'contact', 'month', 'day_of_week', 'poutcome'],
                    prefix = ['job', 'marital', 'education', 'default', 'housing', 'loan',
                               'contact', 'month', 'day_of_week', 'poutcome'])

# Label Encoding
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

Y = encoder.fit_transform(Y)
```

X에 대한 info를 보면 실수형 데이터가 object로 저장되어있다.

```
X.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 62 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   age              41188 non-null   object 
 1   campaign         41188 non-null   object 
 2   pdays             41188 non-null   object 
 3   previous          41188 non-null   object 
 4   emp.var.rate      41188 non-null   object 
 5   cons.price.idx    41188 non-null   object 
 6   cons.conf.idx     41188 non-null   object 
 7   euribor3m         41188 non-null   object 
 8   nr.employed       41188 non-null   object 
 9   job_admin.        41188 non-null   uint8  
 10  job_blue-collar  41188 non-null   uint8  
 11  job_entrepreneur 41188 non-null   uint8  
 12  job_housemaid    41188 non-null   uint8  
 13  job_management    41188 non-null   uint8  
 14  job_retired       41188 non-null   uint8  
 15  job_self-employed 41188 non-null   uint8  
 16  job_services      41188 non-null   uint8  
 17  job_student        41188 non-null   uint8  
 18  job_technician    41188 non-null   uint8  
 19  job_unemployed    41188 non-null   uint8  
 20  job_unknown        41188 non-null   uint8  
 21  marital_divorced  41188 non-null   uint8  
 22  marital_married   41188 non-null   uint8  
 23  marital_single     41188 non-null   uint8  
 24  marital_unknown    41188 non-null   uint8  
 25  education_basic.4y 41188 non-null   uint8  
 26  education_basic.6y 41188 non-null   uint8  
 27  education_basic.9y 41188 non-null   uint8  
 28  education_high.school 41188 non-null   uint8  
 29  education_illiterate 41188 non-null   uint8  
 30  education_professional.course 41188 non-null   uint8  
 31  education_university.degree 41188 non-null   uint8  
 32  education_unknown   41188 non-null   uint8
```

```

33 default_no          41188 non-null  uint8
34 default_unknown     41188 non-null  uint8
35 default_yes         41188 non-null  uint8
36 housing_no          41188 non-null  uint8
37 housing_unknown      41188 non-null  uint8
38 housing_yes          41188 non-null  uint8
39 loan_no              41188 non-null  uint8
40 loan_unknown          41188 non-null  uint8
41 loan_yes              41188 non-null  uint8
42 contact_cellular     41188 non-null  uint8
43 contact_telephone     41188 non-null  uint8
44 month_apr             41188 non-null  uint8
45 month_aug             41188 non-null  uint8
46 month_dec             41188 non-null  uint8
47 month_jul             41188 non-null  uint8
48 month_jun             41188 non-null  uint8
49 month_mar             41188 non-null  uint8
50 month_may             41188 non-null  uint8
51 month_nov             41188 non-null  uint8
52 month_oct             41188 non-null  uint8
53 month_sep             41188 non-null  uint8
54 day_of_week_fri       41188 non-null  uint8
55 day_of_week_mon       41188 non-null  uint8
56 day_of_week_thu       41188 non-null  uint8
57 day_of_week_tue       41188 non-null  uint8
58 day_of_week_wed       41188 non-null  uint8
59 poutcome_failure      41188 non-null  uint8
60 poutcome_nonexistent   41188 non-null  uint8
61 poutcome_success      41188 non-null  uint8
dtypes: object(9), uint8(53)
memory usage: 4.9+ MB

```

object형 feature를 모두 float으로 바꾼다.

```

col = X.select_dtypes("object").columns
for name in col :
    X[name] = X[name].astype(float)

```

마지막으로 train/test data를 분리한다.

```

# train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y,
                                                    shuffle = True,
                                                    test_size = 0.2,
                                                    random_state=1)

```

### c) model fitting

#### c-1) Decision Tree

```

# DT
from sklearn.tree import DecisionTreeClassifier

mytree = DecisionTreeClassifier(criterion = "gini")
mytree.fit(X_train, y_train)

```

```
DecisionTreeClassifier()
```

### c-2) Gradient Boosting

```
# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier

mygbt = GradientBoostingClassifier(learning_rate = 0.1)
mygbt.fit(X_train, y_train)

GradientBoostingClassifier()
```

### c-3) XGBoost

```
# XGBoost
import xgboost

myxgbt = xgboost.XGBClassifier(use_label_encoder = False)
myxgbt.fit(X_train, y_train)

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

### c-4) Bagging

```
# Bagging
from sklearn.ensemble import BaggingClassifier

mybag = BaggingClassifier(DecisionTreeClassifier(),
                          n_estimators = 500,
                          max_samples = 100, bootstrap = True,
                          random_state = 1)
mybag.fit(X_train, y_train)

BaggingClassifier(base_estimator=DecisionTreeClassifier(), max_samples=100,
                  n_estimators=500, random_state=1)
```

### c-5) Random Forest

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier

myrf = RandomForestClassifier(n_estimators = 500, max_depth = 5, random_state=1)
myrf.fit(X_train, y_train)

RandomForestClassifier(max_depth=5, n_estimators=500, random_state=1)
```

## d) Test model

```

# function evaluating accuracy
def evaluate_model(model, x_test, y_test):
    from sklearn import metrics
    import pandas as pd

    # Predict test data
    y_pred = model.predict(x_test)

    # calculate accuracy
    acc = metrics.accuracy_score(y_test, y_pred)

    # Display confusion matrix
    mtx = metrics.confusion_matrix(y_test, y_pred)
    label = ['no', 'yes']
    table = pd.DataFrame(mtx, columns = label, index = label)

    return table, acc

# function printing result
def return_fun(mymodel, X_test, y_test):
    print("====")
    print(f'{mymodel} 결과')
    print("====")
    print(f'  confusion matrix')
    print(f'{evaluate_model(mymodel, X_test, y_test)[0]}')
    print(f'  accuracy')
    print(f'{evaluate_model(mymodel, X_test, y_test)[1]}')
    print('\n')

# Decision Tree
return_fun(mytree, X_test, y_test)

# Gradient Boosting
return_fun(mygbt, X_test, y_test)

# XGBoost
return_fun(myxgbt, X_test, y_test)

# Bagging
return_fun(mybag, X_test, y_test)

# Random Forest
return_fun(myrf, X_test, y_test)

=====
DecisionTreeClassifier() 결과
=====

confusion matrix
    no   yes
no    6569   725
yes    616   328
accuracy
0.8372177713037144

```

```
=====
GradientBoostingClassifier() 결과
=====

confusion matrix
no yes
no 7178 116
yes 704 240
accuracy
0.9004612770089827

=====

XGBClassifier(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...) 결과
=====

confusion matrix
no yes
no 7111 183
yes 686 258
accuracy
0.8945132313668366

=====

BaggingClassifier(base_estimator=DecisionTreeClassifier(), max_samples=100,
                  n_estimators=500, random_state=1) 결과
=====

confusion matrix
no yes
no 7219 75
yes 756 188
accuracy
0.8991260014566642

=====

RandomForestClassifier(max_depth=5, n_estimators=500, random_state=1) 결과
=====

confusion matrix
no yes
no 7231 63
yes 780 164
accuracy
0.8976693372177713
```

마지막으로 accuracy를 시각화하여 비교한다.

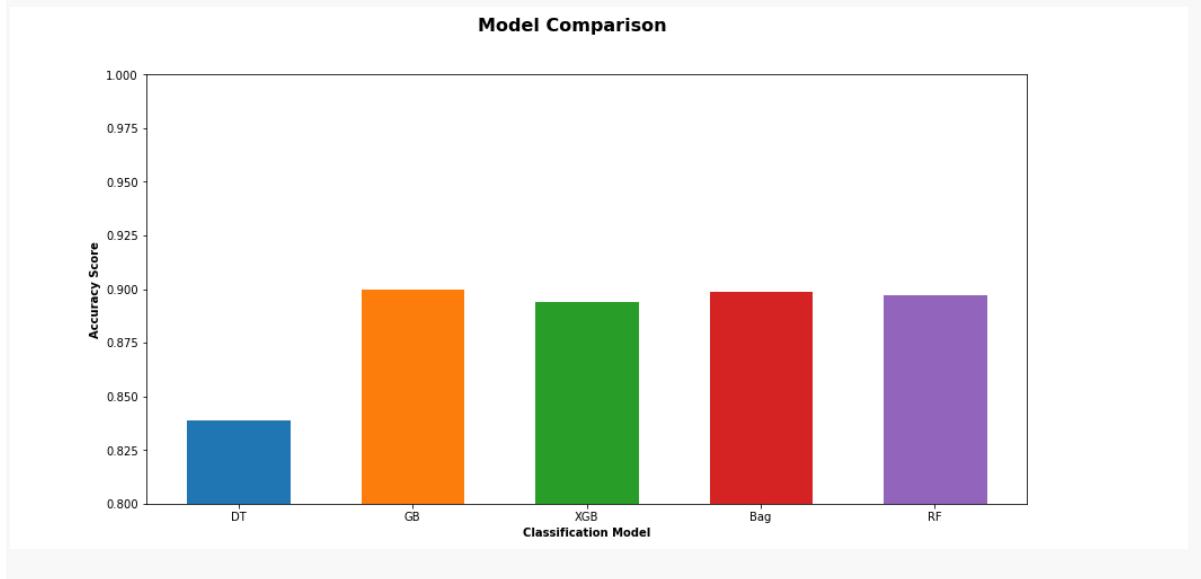
```
import numpy as np
import matplotlib.pyplot as plt

# Initialize figure
fig, ax1 = plt.subplots(1, 1)
fig.suptitle('Model Comparison', fontsize=16, fontweight='bold')
fig.set_figheight(7)
fig.set_figwidth(14)
fig.set_facecolor('white')

## set bar size
barWidth = 0.6

## Make the plot
ax1.bar(0, evaluate_model(mytree, X_test, y_test)[1], width=barWidth, edgecolor='white')
ax1.bar(1, evaluate_model(mygbt, X_test, y_test)[1], width=barWidth, edgecolor='white')
ax1.bar(2, evaluate_model(myxgbt, X_test, y_test)[1], width=barWidth, edgecolor='white')
ax1.bar(3, evaluate_model(mybag, X_test, y_test)[1], width=barWidth, edgecolor='white')
ax1.bar(4, evaluate_model(myrf, X_test, y_test)[1], width=barWidth, edgecolor='white')

## Configure x and y axis
ax1.set_xlabel('Classification Model', fontweight='bold')
labels = ['DT', 'GB', 'XGB', 'Bag', 'RF']
ax1.set_xticklabels(labels)
ax1.set_ylabel('Accuracy Score', fontweight='bold')
ax1.set_ylim(0.8, 1.0)
plt.show()
```



Gradient Boosting의 accuracy는 90%로 5 가지 모델 중 performance가 가장 좋다.