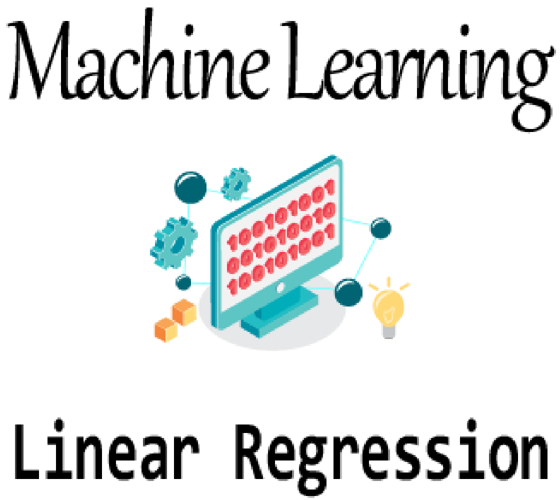
CUAI BASIC 스터디 5조

2022.05.09

발표자 :

목차: 5장 회귀



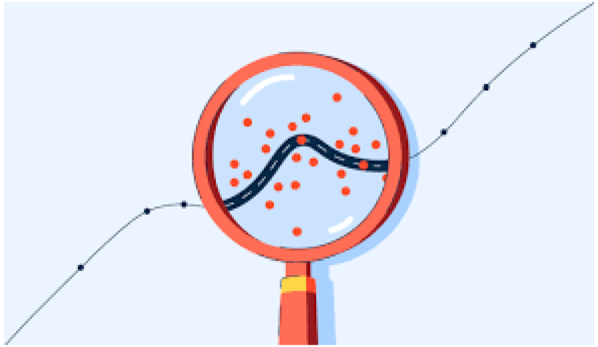
01. 회귀 소개

회귀(Regression)

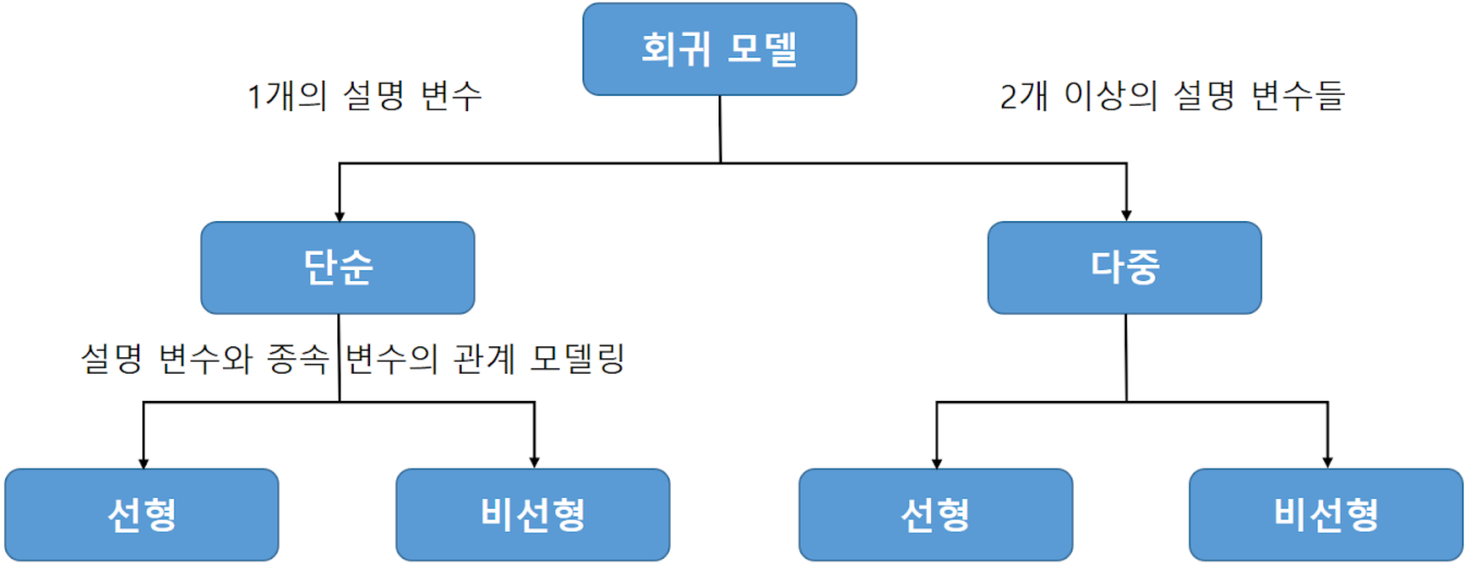
여러 개의 **독립변수**와 한 개의 **종속변수** 간의 상관관계를 모델링하는 기법을 통칭

$$Y = W_1 * X_1 + W_2 * X_2 + W_3 * X_3 + \cdots + W_n * X_n$$

$W_1, W_2, W_3, \cdots, W_n$: 독립변수의 값에 영향을 미치는 **회귀 계수** (Regression Coefficients)



회귀 유형 구분

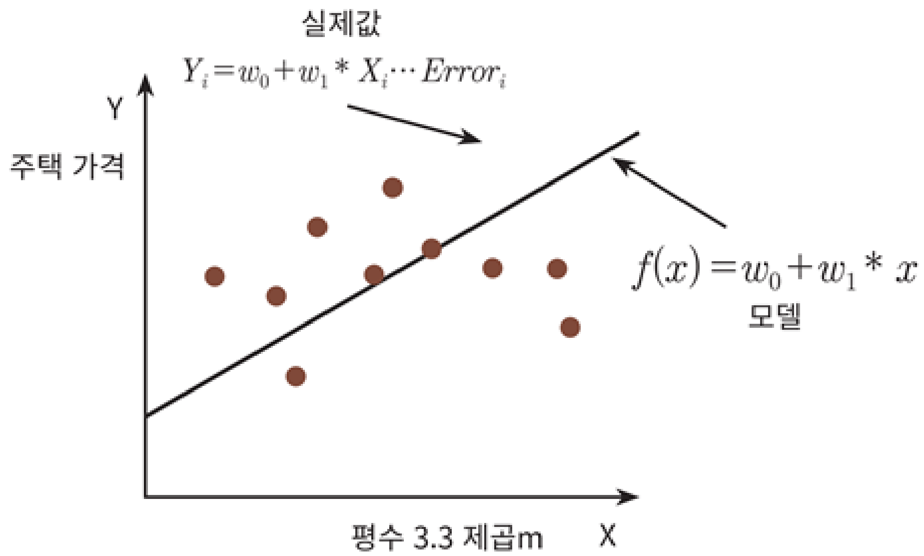


독립변수 개수: 단일 회귀(1개), 다중 회귀(여러 개)
회귀 계수의 결합: 선형 회귀, 비선형 회귀

02. 단순 선형 회귀를 통한 회귀 이해

단순 선형 회귀

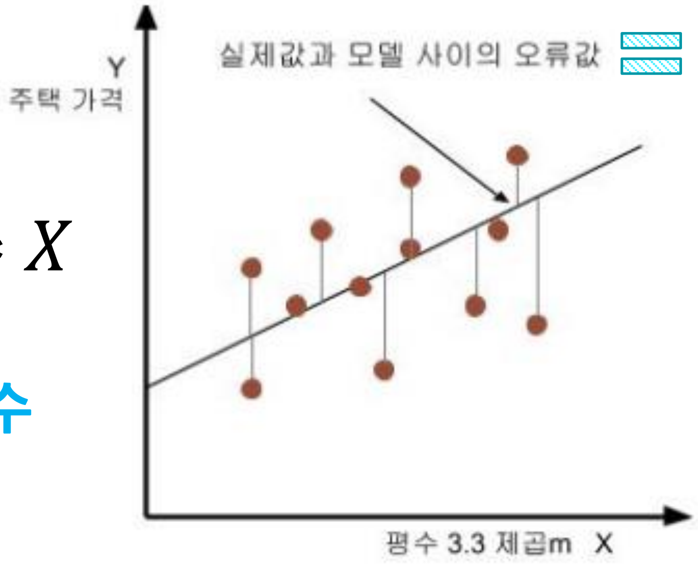
독립변수도 하나, 종속변수도 하나인 선형 회귀



회귀

$$\hat{Y} = w_0 + w_1 * X$$

w_0 w_1 : 회귀 계수
’(intercept)



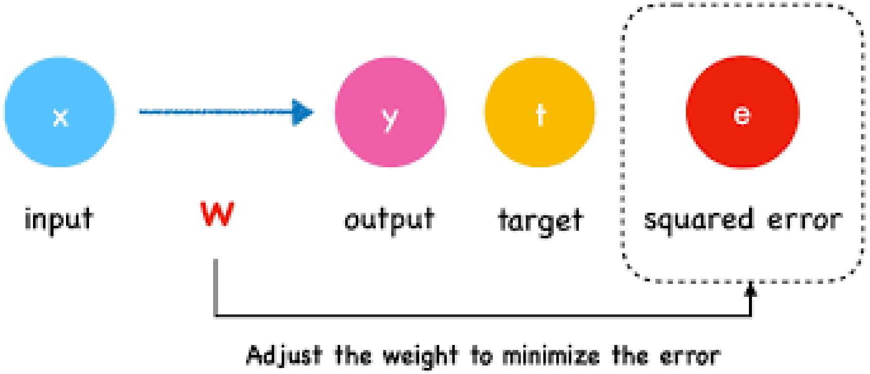
= 잔차

최적의 회귀모델을 만든다
= 전체 데이터의 잔차(오류 값) 합이 최소가 되는 모델을 만든다

Mean Absolute Error: 오류 합을 계산할 때 절댓값을 취해서 더함
Residual Sum of Square: 오류 값의 제곱을 구해서 더하는 방식

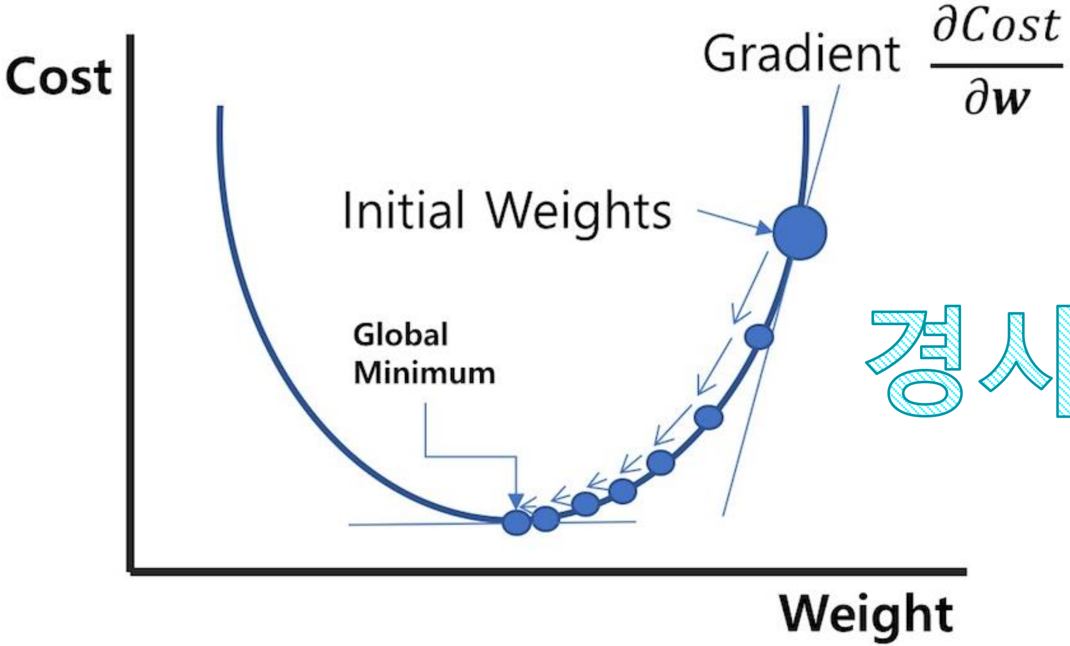
$$\boxed{RSS(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (w_0 + w_1 * xi)\right)^2}$$

비용(Cost) 함수



03. 비용 최소화하기 - 경사하강법 소개

점진적으로 반복적인 계산을 통해 W 파라미터 값을 **업데이트**하면서
오류 값이 최소가 되는 W 파라미터는 구하는 방식



경사 하강법

$$\boxed{R(w) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - (w_0 + w_1 * xi)\right)^2}$$

η = 학습률(Learning Rate)

$$w_{1,new} = w_{1,old} - \eta\frac{dLoss(w)}{dw_1} = w_{1,old} + \eta\left(\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)\right)$$

$$w_{0,new} = w_{0,old} - \eta\frac{dLoss(w)}{dw_0} = w_{0,old} + \eta\left(\frac{2}{N}\sum_{i=1}^{N} \text{실제값}_i - \text{예측값}_i\right)$$

$$\hat{Y} \qquad\qquad X_{mat}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdots\cdots \\ y_n \end{bmatrix} = \begin{array}{cccc} \text{Feature 1} & \text{Feature 2} & \cdots\cdots & \text{Feature m} \\ \end{array} \begin{pmatrix} x_{11} & x_{12} & \cdots\cdots & x_{1m} \\ x_{21} & x_{22} & \cdots\cdots & x_{2m} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ x_{n1} & x_{n2} & \cdots\cdots & x_{nm} \end{pmatrix} \underset{\text{내적}}{*} \begin{bmatrix} w_1 & w_2 & \cdots\cdots & w_m \end{bmatrix}^T + w_0$$

04. 사이킷런 LinearRegression

LinearRegression: 선형 모델 중 규제가 적용되지 않은 선형회귀를 사이킷런에서 구현한 클래스

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

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

X_train , X_test , y_train , y_test = train_test_split(X_data , y_target ,test_size=0.3, random_state=156)

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

Linear_model 모듈

Ordinary Least Square 방식

다중 공선성 (multi-collinearity) : 피처 간의 상관관계가 매우 높은 경우 분산이 매우 커져서 오류에 매우 민감해지는 문제

회귀 평가 지표

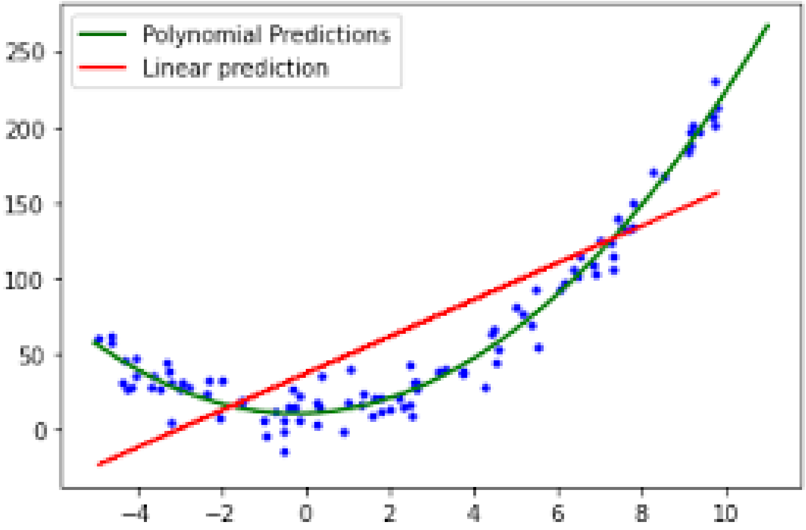
- MSE (Mean Squared Error) = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- MAE (Mean absolute error) = $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- RMSE (Root Mean Squared Error) = $\sqrt{MSE}$
- R-squared (Coefficient of determination) = $1 - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y}_i)^2} = 1 - \frac{SSE}{SST} = \frac{SSR}{SST}$
- 참고
  - SSE(Sum of Squares `Error`, 관측치와 예측치 차이): $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - SSR(Sum of Squares due to `Regression`, 예측치와 평균 차이): $\sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2$
  - SST(Sum of Squares `Total`, 관측치와 평균 차이): $\sum_{i=1}^{n}(y_i - \bar{y}_i)^2$, SSE + SSR

MSE: mean_squared-error(실제값, 예측값, squared=True)
RMSE: mean_squared-error(실제값, 예측값, squared=False)

05. 다항 회귀와 과(대)적합/과소적합 이해

다항 (Polynomial) 회귀

독립변수의 단항식이 아닌, 고차 방정식과 같은 다항식으로 표현되는 회귀
회귀 계수는 선형! 다항 회귀는 선형 회귀



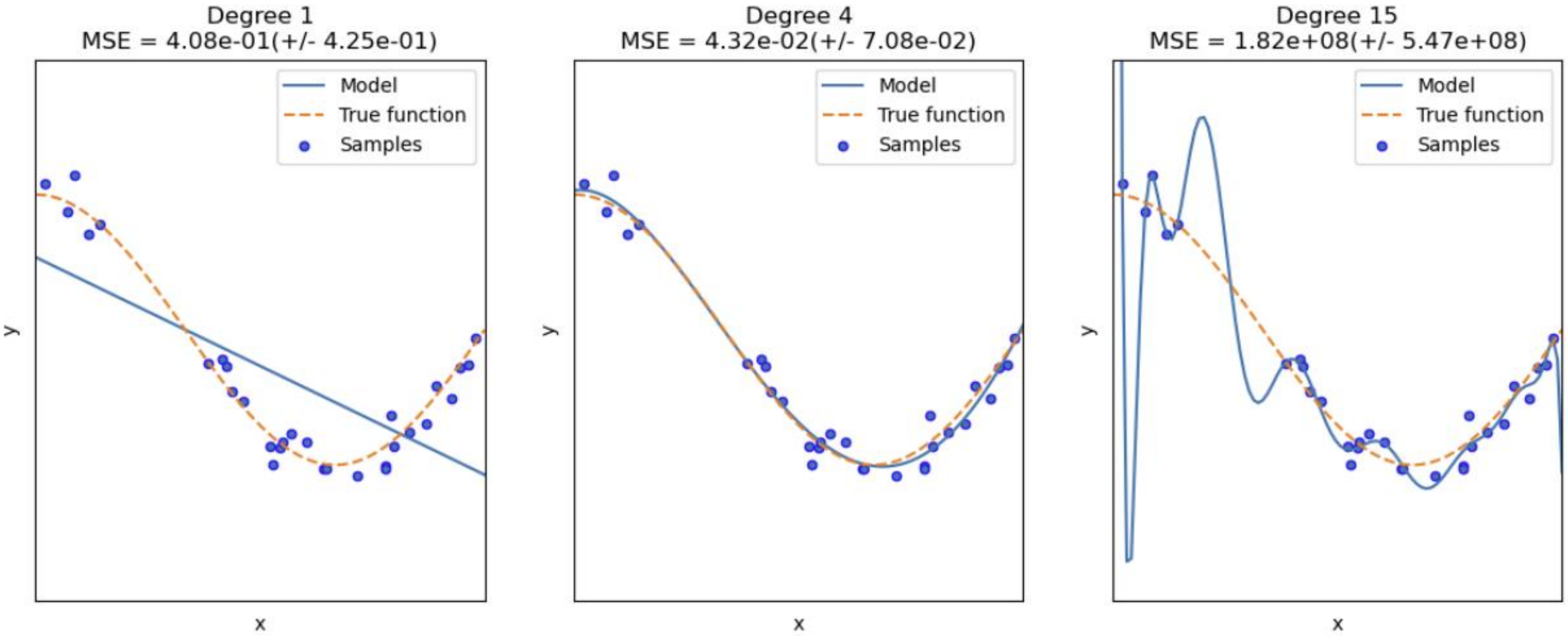
사이킷런: 비선형 함수를 선형 모델에 적용시키는 방법 사용

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

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

# Pipeline 객체로 Streamline 하게 Polynomial Feature변환과 Linear Regression을 연결
model = Pipeline([('poly', PolynomialFeatures(degree=3)),
                  ('linear', LinearRegression())])
X = np.arange(4).reshape(2,2)
y = polynomial_func(X)

model = model.fit(X, y)
print('Polynomial 회귀 계수\n', np.round(model.named_steps['linear'].coef_, 2))
```

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

PolynomialFeatures 클래스: degree 파라미터를 통해 입력 받은 단항식 피처를 degree에 해당하는 다항식 피처로 변환

다항 회귀의 문제점



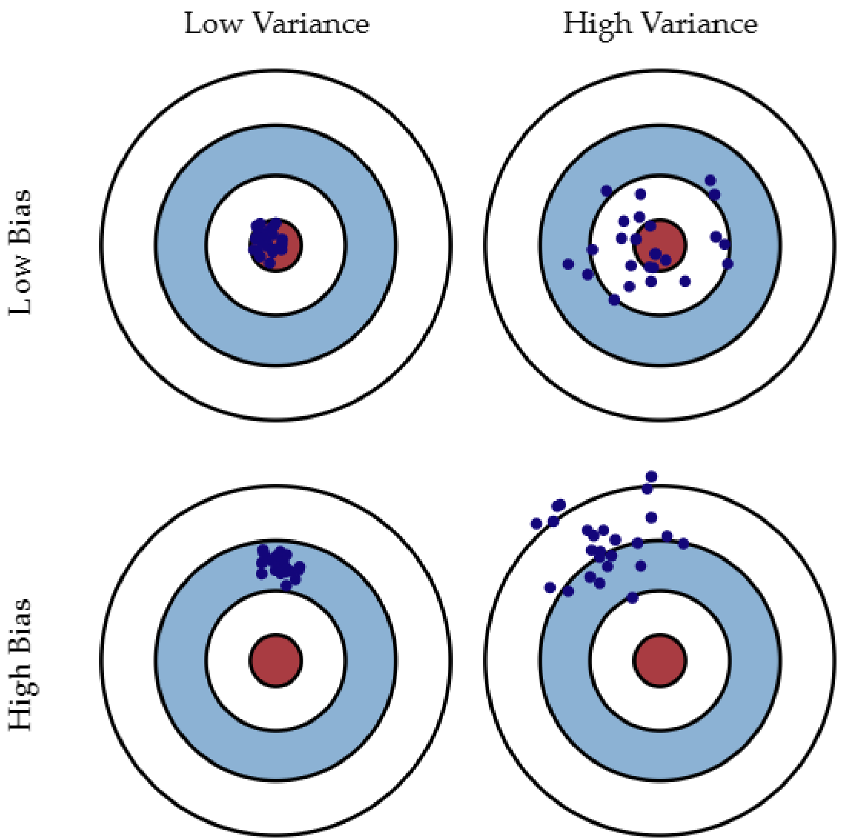
차수(degree)를 높일수록 학습데이터에만 맞춘 학습이 이루어져
테스트 환경에서는 오히려 예측 정확도 떨어짐 >> 과적합 문제

이론

편향(Bias)과 분산(Variance)

저편향
저분산

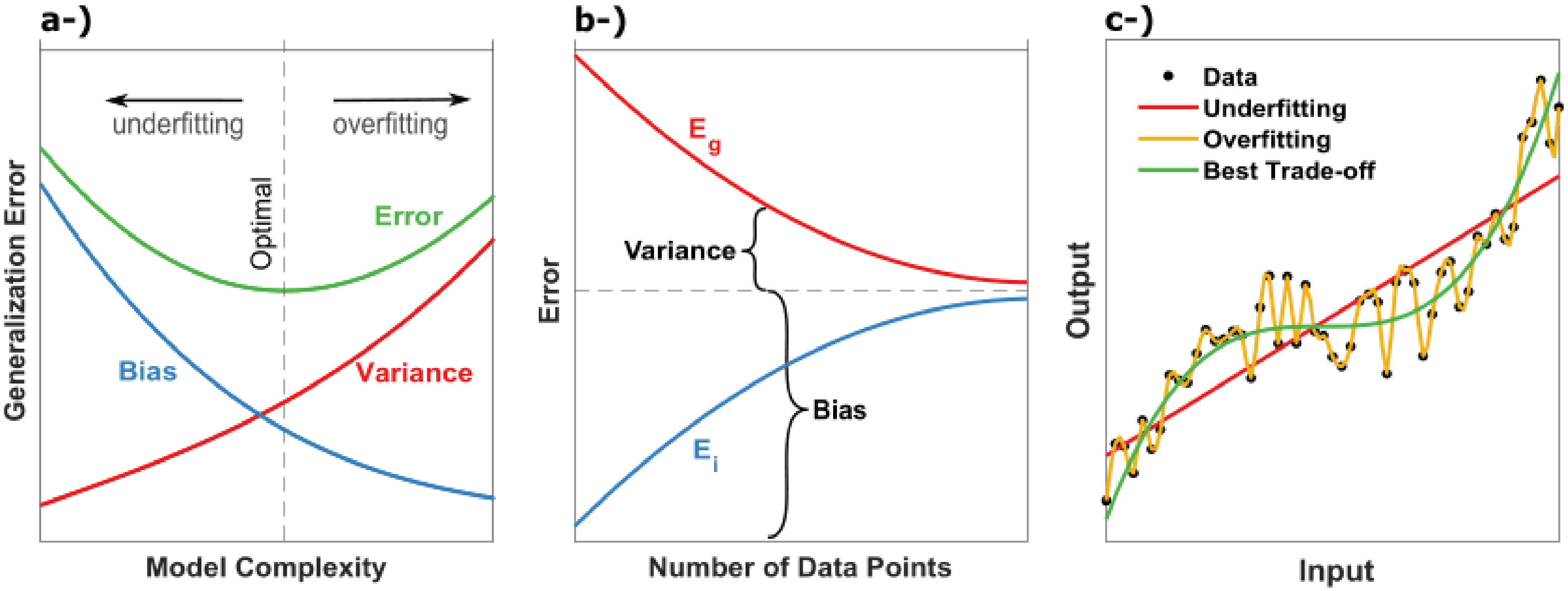
고편향
저분산



저편향
고분산

고편향
고분산

편향(Bias) - 분산(Variance) Trade Off



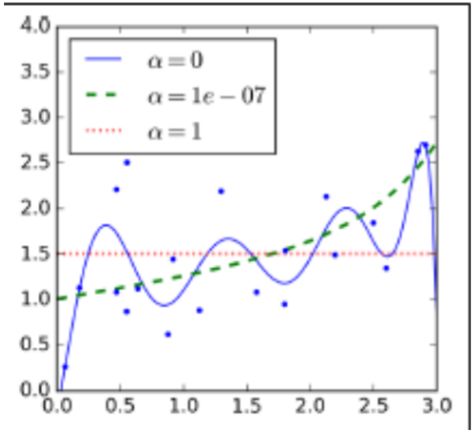
일반적으로, 편향과 분산은 한쪽이 높으면 한쪽이 낮아지는 관계

과소적합: 높은 변향, 낮은 분산
과적합: 높은 분산, 낮은 편향

06. 규제 선형 모델 - 릿지, 라쏘, 엘라스틱넷

	• 릿지 회귀	• 라쏘 회귀
비용함수	$Min(RSS(W)+\alpha*\|\|W_2\|\|^2)$	$Min(RSS(W)+\alpha*\|\|W_1\|\|)$
규제 종류	L2 규제	L1 규제
회귀계수	—	0으로 만들 수 있음

• 엘라스틱넷 회귀
$Min(RSS(W)+\alpha_2*\|\|W_2\|\|^2+\alpha_1*\|\|W_1\|\|)$
L1, L2 규제 결합
급격한 변동 막을 수 있음



07. 로지스틱 회귀

1.정의:

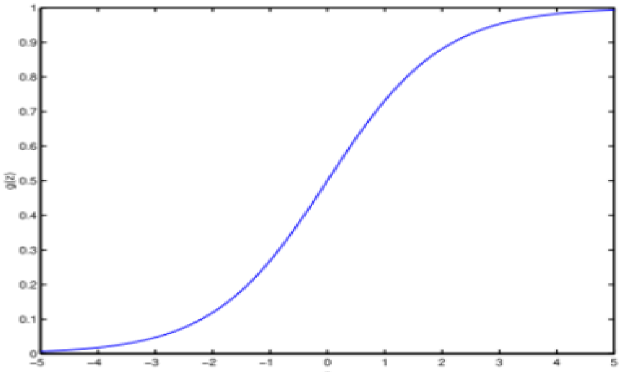
- 로지스틱 회귀는 선형 회귀와 유사하지만, 이진 분류 문제에 적용됩니다.
- 종속 변수가 범주형이며, 주로 0과 1의 값을 갖습니다.

2.로지스틱 함수 (시그모이드 함수):

- 로지스틱 함수는 입력 값을 0과 1 사이의 확률로 변환합니다.
- 시그모이드 함수라고도 불립니다.

3.확률 추정:

- 로지스틱 회귀는 각 클래스에 속할 확률을 추정합니다.
- 추정된 확률을 기준으로 분류 결정을 내립니다.



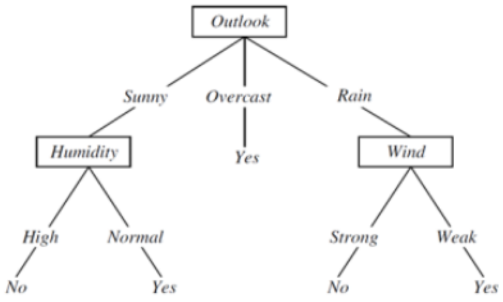
08. 회귀 트리

1. 정의:

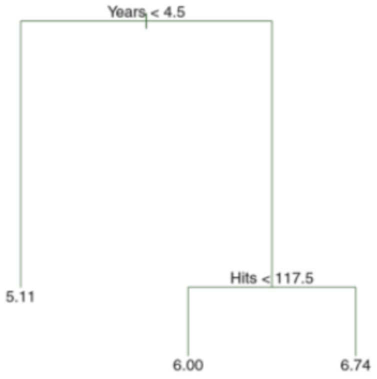
1. 결정 트리 알고리즘을 확장하여 연속적인 값을 예측하는 모델입니다.
2. 결정 트리와 비슷한 구조를 가지며, 각 리프 노드에서 예측값을 생성합니다.

2. 예측 방법:

1. 새로운 데이터가 트리를 따라 내려가면서 해당하는 리프 노드에 도달합니다.
2. 리프 노드의 평균 값이 최종 예측값으로 사용됩니다.



분류 트리



회귀 트리

09. 회귀 실습 - 자전거 대여 수요 예측

Playground Prediction Competition

Bike Sharing Demand

Forecast use of a city bikeshare system

Kaggle · 3,242 teams · 8 years ago

```
drop_columns = ['datetime','casual','registered']
bike_df.drop(drop_columns, axis=1,inplace=True)
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

bike_df = pd.read_csv('./bike_train.csv')
print(bike_df.shape)
bike_df.head(3)
```

(10886, 12)

8]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32

```
bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

09. 회귀 실습 - 자전거 대여 수요 예측

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# log 값 변환 시 NaN등의 이슈로 log() 가 아닌 log1p() 를 이용하여 RMSLE 계산
def rmsle(y, pred):
    log_y = np.log1p(y)
    log_pred = np.log1p(pred)
    squared_error = (log_y - log_pred) ** 2
    rmsle = np.sqrt(np.mean(squared_error))
    return rmsle

# 사이킷런의 mean_square_error() 를 이용하여 RMSE 계산
def rmse(y,pred):
    return np.sqrt(mean_squared_error(y,pred))

# MSE, RMSE, RMSLE 를
def evaluate_regr(y,p
    rmsle_val = rmsle
    rmse_val = rmse(y
    # MAE 는 scikit l
    mae_val = mean_ab
    print('RMSLE: {0:
```

```
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.linear_model import LinearRegression , Ridge , Lasso

y_target = bike_df['count']
X_features = bike_df.drop(['count'],axis=1,inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.3, random_state=0)

lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
pred = lr_reg.predict(X_test)

evaluate_regr(y_test ,pred)
```

```
RMSLE: 1.165, RMSE: 140.900, MAE: 105.924
```

09. 회귀 실습 - 자전거 대여 수요 예측

```
coef = pd.Series(lr_reg.coef_, index=X_features.columns)
coef_
sns.b
plt.s
```

```python
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

# 랜덤 포레스트, GBM, XGBoost, LightGBM model 별로 평가 수행
rf_reg = RandomForestRegressor(n_estimators=500)
gbm_reg = GradientBoostingRegressor(n_estimators=500)
xgb_reg = XGBRegressor(n_estimators=500)
lgbm_reg = LGBMRegressor(n_estimators=500)

for model in [rf_reg, gbm_reg, xgb_reg, lgbm_reg]:
    # XGBoost의 경우 DataFrame이 입력 될 경우 버전에 따라 오류 발생 가능. ndarray로 변환.
    get_model_predict(model,X_train.values, X_test.values, y_train.values, y_test.values,is_expm1=True)
```

```
### RandomForestRegressor ###
RMSLE: 0.355, RMSE: 50.377, MAE: 31.238
### GradientBoostingRegressor ###
RMSLE: 0.330, RMSE: 53.322, MAE: 32.734
### XGBRegressor ###
RMSLE: 0.342, RMSE: 51.732, MAE: 31.251
### LGBMRegressor ###
RMSLE: 0.319, RMSE: 47.215, MAE: 29.029
```

wind

hu

w

s

h

worki

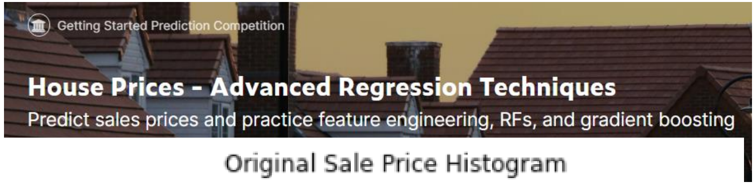
33

93

303

```
RMSLE: 1.017, RMSE: 162.594, MAE: 109.286
```

09. 회귀 실습 - 캐글 주택 가격: 고급 회귀 기법




```python
print('get_dummies() 수행 전 데이터 Shape:', house_df.shape)
house_df_ohe = pd.get_dummies(house_df)
print('get_dummies() 수행 후 데이터 Shape:', house_df_ohe.shape)

null_column_count = house_df_ohe.isnull().sum()[house_df_ohe.isnull().sum() > 0]
print('## Null 피처의 Type :\n', house_df_ohe.dtypes[null_column_count.index])
```

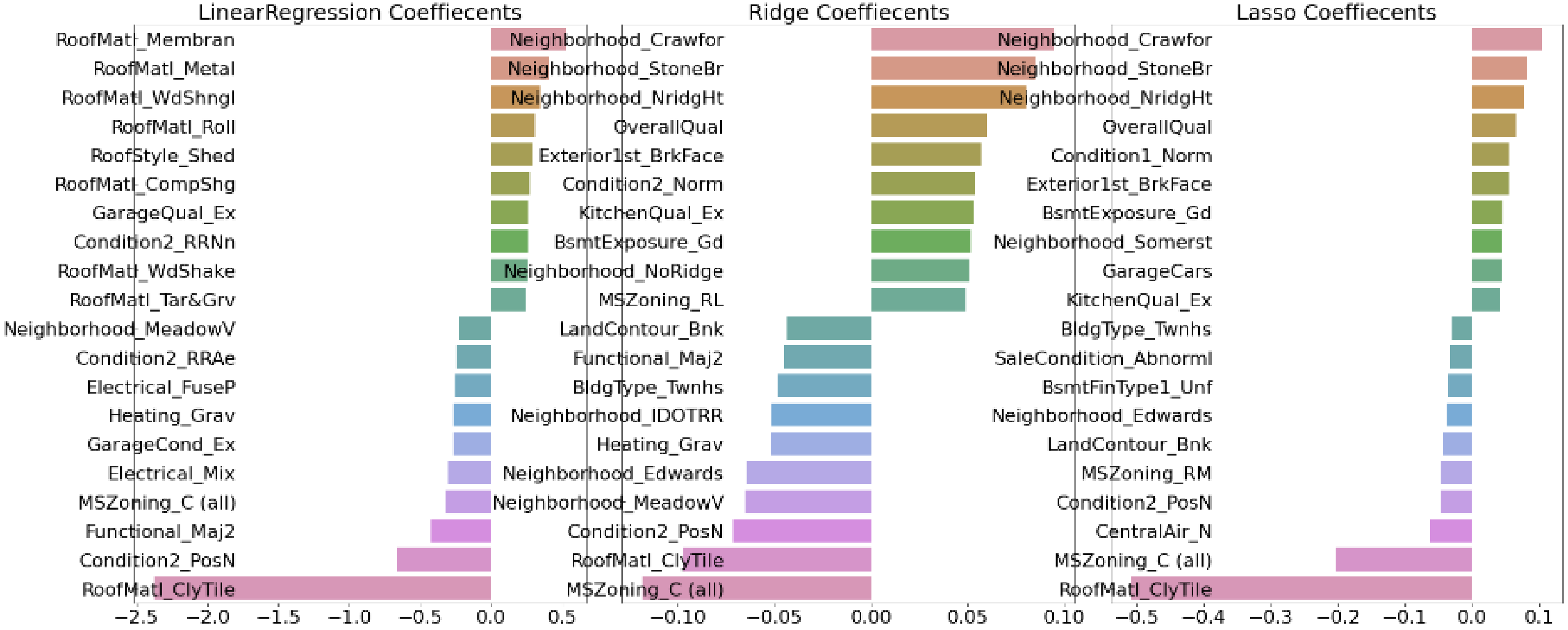
```
get_dummies() 수행 전 데이터 Shape: (1460, 75)
get_dummies() 수행 후 데이터 Shape: (1460, 271)
## Null 피처의 Type :
 Series([], dtype: object)
```

2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	Na

3 rows × 81 columns

09. 회귀 실습 - 캐글 주택 가격: 고급 회귀 기법

```
LinearRegression 로그 변환된 RMSE: 0.132
Ridge 로그 변환된 RMSE: 0.124
Lasso 로그 변환된 RMSE: 0.12
```



09. 회귀 실습 - 캐글 주택 가격: 고급 회귀 기법

```python
# get_stacking_base_datasets( )은 넘파이 ndarray를 인자로 사용하므로 DataFrame을 넘파이로 변환.
X_train_n = X_train.values
X_test_n = X_test.values
y_train_n = y_train.values

# 각 개별 기반(Base)모델이 생성한 학습용/테스트용 데이터 반환.
ridge_train, ridge_test = get_stacking_base_datasets(ridge_reg, X_train_n, y_train_n, X_test_n, 5)
lasso_train, lasso_test = get_stacking_base_datasets(lasso_reg, X_train_n, y_train_n, X_test_n, 5)
xgb_train, xgb_test = get_stacking_base_datasets(xgb_reg, X_train_n, y_train_n, X_test_n, 5)
lgbm_train, lgbm_test = get_stacking_base_datasets(lgbm_reg, X_train_n, y_train_n, X_test_n, 5)
```

```
Ridge  model 시작
	 폴드 세트:  0  시작
	 폴드 세트:  1  시작
	 폴드 세트:  2  시작
	 폴드 세트:  3  시작
	 폴드 세트:  4  시작
Lasso  model 시작
	 폴드 세트:  0  시작
	 폴드 세트:  1  시작
	 폴드 세트:  2  시작
	 폴드 세트:  3  시작
	 폴드 세트:  4  시작
XGBRegressor  model 시작
	 폴드 세트:  0  시작
	 폴드 세트:  1  시작
	 폴드 세트:  2  시작
	 폴드 세트:  3  시작
	 폴드 세트:  4  시작
LGBMRegressor  model 시작
```

```
스태킹 회귀 모델의 최종 RMSE 값은: 0.09799152965189681
```

감사합니다.