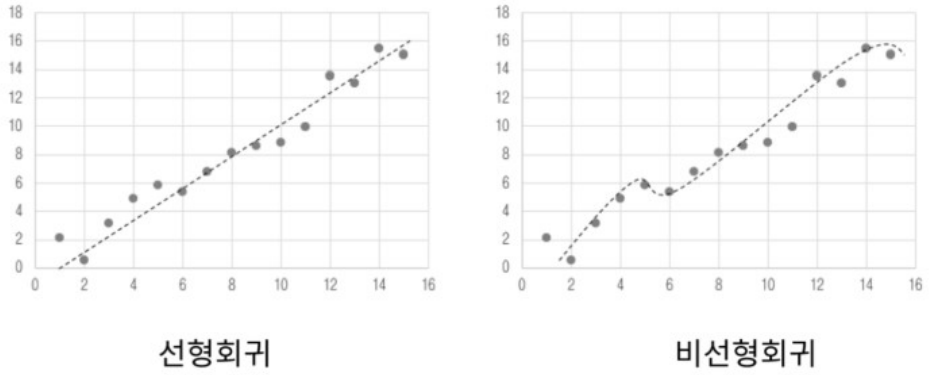
선형 회귀

선형 회귀

- 회귀(regression) : 입력 변수 X에 대해서 연속형 출력 변수 Y를 예측
- 분류(classification) : 입력 변수 X에 대해서 이산형 출력 변수 Y(class)를 예측

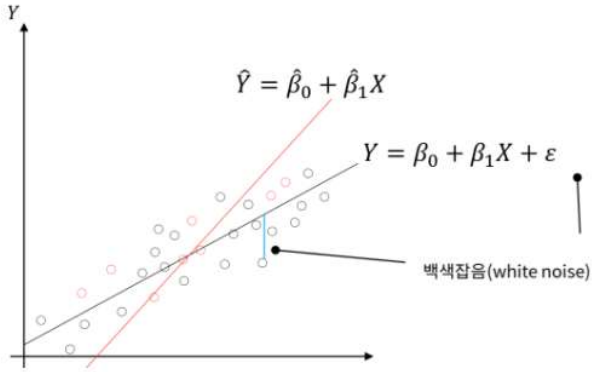
- 회귀 : 입력변수인 X의 정보를 활용하여 출력변수인 Y를 예측하는 방법



- 선형 회귀는 실제값과 예측값의 차이인 오류를 최소로 줄일 수 있는 선형 함수를 찾아서 선형 함수에 독립변수(피처)를 입력해 종속변수(타깃값, 예측값)을 예측하는 것입니다.
- 최적의 선형 함수를 찾기 위해 실제값과 예측 값 사이의 제곱을 회귀 계수 W를 변수로 하는 비용 함수를 만들고 이 비용함수가 최소화되는 W의 값을 찾아 선형 함수를 도출 할 수 있습니다.

선형 회귀

- 단순 선형 회귀는 독립변수도 하나, 종속변수도 하나인 선형 회귀입니다.
- 잔차 : 실제 값과 회귀 모델의 차이에 따른 오류 값
- 최적의 회귀 모델 : 직선과 데이터의 차이가 평균적으로 가장 작아지는 직선 전체 데이터의 잔차(오류 값) 합이 최소가 되는 모델을 만드는 것
- 오류의 합을 계산할 때는 절댓값을 취해서 더하거나 (Mean Absolute Error), 오류 값의 제곱을 구해서 더하는 방식(Rss, Residual Sum of Square)을 취합니다.
- 일반적으로 미분 등의 계산을 편리하게 하기 위해서 RSS(Residual Sum of Square) 방식으로 오류 합을 구합니다.
- Error^2 = RSS
- 회귀에서 RSS는 비용이며 w 변수(회귀 계수)로 구성되는 RSS를 비용함수라고 합니다.



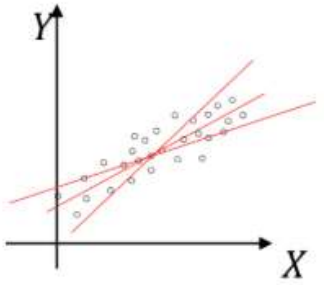
$$SSE = \sum_{i=1}^{n} e_i^2 = e_1^2 + e_2^2 + \cdots + e_n^2$$

- 머신러닝 회귀 알고리즘은 데이터를 계속 학습시키면서 비용함수가 반환하는 값(오류 값)을 지속해서 감소시키고 최종적으로 더 이상 감소하지 않는 최소의 오류 값을 구하는 것
- 비용함수 = 손실함수(loss function)

선형 회귀

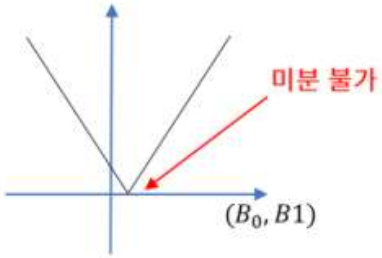
- 굳이 잔차의 제곱합을 최소화 시키는 이유
- 잔차의 합이 0이 되는 해는 무수히 많음 (유일한 해를 찾지 못함)

$$\sum_{i=1}^{n} e_i = e_1 + e_2 + \cdots + e_n = 0$$



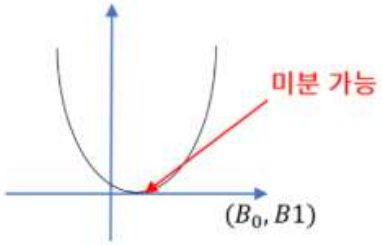
- 잔차의 절대값의 합은 미분이 불가능한 형태

$$\sum_{i=1}^{n} |e_i| = |e_1| + |e_2| + \cdots + |e_n|$$



- 잔차의 제곱 합은 미분이 가능한 형태로 유일한 해를 찾을 수 있음

$$SSE = \sum_{i=1}^{n} e_i^2 = e_1^2 + e_2^2 + \cdots + e_n^2$$

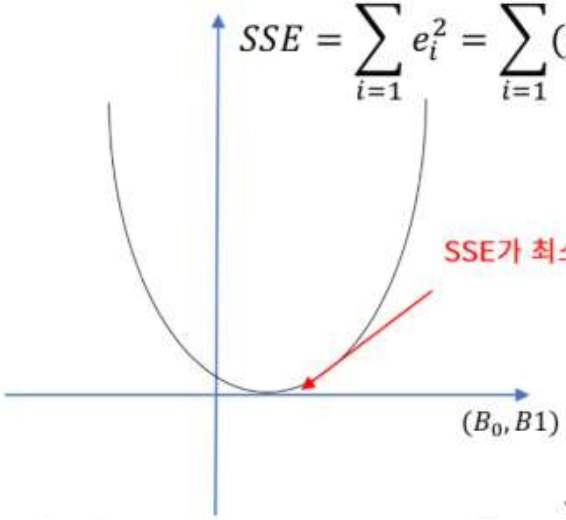


선형 회귀

회귀 계수의 추정

SSE $\hat{\beta}_0$과 $\hat{\beta}_1$로 편미분하여 연립방정식을 푸는 방법(Least Square Method)

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$



$$\frac{\partial L}{\partial \beta_0} = -2\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) = 0$$

$$\frac{\partial L}{\partial \beta_1} = -2\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)x_i = 0$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

선형 회귀

➢ 회귀 유형

- 일변량(Univariate) - 오직 하나의 양적 독립변수(설명변수)
- 다변량(Multivariate) - 두 개 이상의 양적 독립변수(설명변수)
- 단순(Simple) - 오직 하나의 종속변수(반응변수)
- 다중(Multiple) - 두 개 이상의 종속변수(반응변수)
- 선형(Linear) - 데이터에 대하여 가능한 변환을 취한 후, 모든 계수들이 방정식에 선형적으로 삽입되어 있음.
- 비선형(Nonlinear) - 종속변수(반응변수)와 일부 독립변수들의 관계가 비선형이거나 일부 계수들이 비선형적으로 나타남. 계수들을 선형적으로 나타나게 하는 어떤 변환도 가능하지 않음.
- 분산분석(ANOVA) - 모든 독립변수들이 질적 변수임
- 공분산분석(ANCOVA) - 어떤 독립변수들은 양적변수이고 다른 독립변수들은 질적변수임
- 로지스틱(Logistic) - 종속변수(반응변수)가 질적변수임.

단순 선형 회귀분석 : 변수가 1개인 경우

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

다중 선형 회귀분석 : 변수가 여러개인 경우

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \cdots + \hat{\beta}_p X_p$$

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

선형 회귀

- 선형 회귀는 실제 값과 예측값가의 차이(오류의 제곱의 값)를 최소화하는 직선형 회귀선을 최적화하는 방법
- 규제는 선형회귀의 과적합 문제를 해결하기 위해서 회귀 계수에 페널티 값을 적용하는 것
- 규제 방법에 따른 선형 회귀 모델 유형

일반 선형 회귀	예측값과 실제 값의 RSS(Residual Sum of Square)를 최소화할 수 있도록 회귀 계수를 최적화하며, 규제를 적용하지 않은 모델
릿지(Ridge)	선형 회귀에 L2 규제를 추가한 회귀 모델 L2 규제는 상대적으로 큰 회귀 계수 값의 예측 영향도를 감소시키기 위해서 회귀 계수값을 더 작게 만드는 규제 모델 L2 규제는 회귀 계수 값의 크기를 줄입니다.
랏쏘(Lasso)	선형 회귀에 L1 규제를 추가한 회귀 모델 L1 규제는 예측 영향력이 작은 피처의 회귀 계수를 0으로 만들어 회귀 예측 시 피처가 선택되지 않게 하는 것 L1은 피처 선택 기능
엘라스틱넷(ElasticNet)	L2, L1 규제를 함께 결합한 모델 피처가 많은 데이터 세트에서 적용되며, L1 규제로 피처의 개수를 줄임과 동시에 L2 규제로 계수 값의 크기를 조정합니다.
로지스틱 회귀 (Logistic Regression)	분류에 사용되는 회귀 모델 이진 분류, 희소 영역의 분류등 텍스트 분류와 같은 영역에서 뛰어난 예측 성능을 보입니다.

선형 회귀

- 경사하강법 : 비용최소화하기
  - W 파라미터의 개수가 적다면 고차원 방정식으로 비용함수가 최소가 되는 W 변숫값을 도출할 수 있겠지만, W파라미터가 많으면 고차원 방정식을 동원하더라도 해결하기가 어렵습니다.
  - 경사하강법은 고차원 방정식에 대한 문제를 해결해 주면서 비용 함수 RSS를 최소화하는 방법을 직관적으로 제공
  - 반복적으로 W 파라미터 값을 업데이트하면서 오류 값이 최소가 되는 W 파라미터를 구하는 방식
  - 경사하강법은 비용함수의 반환 값 (예측값과 실제값의 차이)이 작아지는 방향성을 가지고 W파라미터를 지속해서 보정해 나갑니다.
  - 가속도가 계속 증가하면서 속도가 증가하고, 더 이상 가속도가 증가하지 않으면 최고 속도이며 가속도가 마이너스(-)가 되면서 속도가 떨어지집니다.
  - 가속도의 값은 속도의 미분으로 구할 수 있습니다
  - 속도와 같은 포물선 형태의 2차 함수의 최저점은 해당 2차 함수의 미분값인 1차 함수의 기울기가 가장 최소일 때입니다.
  - 경사하강법은 최초 w에서부터 미분을 적용한 뒤 이 미분값이 계속 감소하는 방향으로 순차적으로 w를 업데이트합니다.
  - 더 이상 미분된 1차 함수의 기울기가 감소하지 않는 지점을 비용 함수가 최소인 지점으로 간주하고 그때의 w를 반환합니다.
  - gradient_descent()
  - gradient_descent_steps()
  - 경사하강법은 모든 학습 데이터에 대해 반복적으로 비용함수 최소화를 위한 값을 업데이트하기 때문에 수행 시간이 매우 오래 걸린다

선형 회귀

- 경사하강법 : 비용최소화하기
  - 실전에서는 확률적 경사하강법(Stochastic Gradient Descent)를 이용합니다.
  - 확률적 경사하강법(Stochastic Gradient Descent)은 전체 입력 데이터로 w가 없데이트되는 값을 계산하는 것이 아니라 일부 데이터만 이용해 w가 업데이트되는 값을 계산하므로 경사 하강법에 비해 빠른 속도를 보장합니다.
  - 대용량의 데이터의 경우 대부분 확률적 경사하강법이나 미니 배치 확률적 경사 하강법을 이용해 최적 비용 함수를 도출합니다.
  - stochastic_gradient_descent()

  - 사이킷런의 linear_models 모듈 LinearRegression
  - 예측값과 실제 값의 RSS(Residual Sum of Squares)를 최소화해 OLS(Ordinary Least Squares) 추정 방식으로 구현한 클래스입니다.
  - fit()메서드로 X, y 배열을 입력을 받으면 회귀 계수(Coefficient)인 W를 coef_속성에 저장합니다.
  - fit_intercept : 디폴트 값 True, intercept(절편)값을 계산할 것인지 말것인지를 지정
  - normalize : 디폴트 값 False, fit_intercept 가 False인 경우 이 파라미터는 무시됩니다.
  - coef_ : 회귀 계수가 배열 형태로 저장하는 속성
  - intercept_

선형 회귀

- 경사하강법 : 비용최소화하기
  - **MAE (Mean Absolute Error)** - 실제 값과 예측 값의 차이를 절대값으로 변환해 평균한 것, metrics.mean_absolute_error , neg_mean_absolute_error
  - **MSE (Mean Squared Error)** - 실제 값과 예측 값의 차이를 제곱해 평균한 것 , metrics.mean_squared_error , neg_mean_squared_error
  - **RMSE (Root Mean Squared Error)** - MSE 값은 오류의 제곱을 구하므로 실제 오류 평균보다 더 커지는 특성이 있으므로 MSE에 루트를 씌운 것
  - **R^2** - 분산 기반으로 예측 성능을 평가합니다. 실제 값의 분산 대비 예측값의 분산 비율을 지표로 하여 1에 가까울수록 예측 정확도가 높습니다. metrics.r2_score , r2

  - 사이킷런의 Scoring 함수가 score 값이 클수록 좋은 평가 결과로 자동 평가합니다.
  - 실제 값과 예측 값의 오류 차이를 기반으로 하는 회귀 평가 지표의 경우 값이 커지면 오히려 나쁜 모델이라는 의미이므로 이를 사이킷런의 Scoring 함수에 일반적으로 반영하려면 보정이 필요합니다.
  - -1을 원래의 평가 지표 값에 곱해서 음수를 만들어 작은 오류 값이 더 큰 숫자로 인식하게 합니다.
  - neg_mean_absolute_error = -1 * metrics.mean_absolute_error()

선형 회귀

- 회귀분석 - 보스턴 집값 예측
  - 예측(prediction)문제 - 특정한 입력변수값을 사용하여 출력변수의 값을 계산하는 것
  - 출력변수의 값이 연속값인 문제를 회귀(regression) 또는 회귀분석(regression analysis) 문제라고 합니다

보스턴 주택 가격 데이터 독립변수	종속변수
CRIM: 범죄율	보스턴 506개 타운의 1978년 주택 가격 중앙값 (단위 1,000 달러)
INDUS: 비소매상업지역 면적 비율	
NOX: 일산화질소 농도	
RM: 주택당 방 수	
LSTAT: 인구 중 하위 계층 비율	
B: 인구 중 흑인 비율	
PTRATIO: 학생/교사 비율	
ZN: 25,000 평방피트를 초과 거주지역 비율	
CHAS: 찰스강의 경계에 위치한 경우는 1, 아니면 0	
AGE: 1940년 이전에 건축된 주택의 비율	
RAD: 방사형 고속도로까지의 거리	
DIS: 직업센터의 거리	
TAX: 재산세율	

선형 회귀

➢ 회귀 분석 - 보스턴 집값 예측

```
from sklearn.datasets import load_boston
import matplotlib.pyplot as plt
import seaborn as sns

boston = load_boston()
dir(boston)

dfX = pd.DataFrame(boston.data, columns=boston.feature_names)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

df = pd.concat([dfX, dfy], axis=1)
df.tail()

sns.pairplot(df[["MEDV", "RM", "AGE", "CHAS"]])
plt.show()
#종속변수인 집값(MEDV)과 방 개수(RM), 노후화 정도(AGE)와 어떤 관계를 가지는지 알 수 있다.
#방 개수가 증가할 수록 집값은 증가하는 경향이 뚜렷하다.
#노후화 정도와 집값은 관계가 없어 보인다.
```

선형 회귀

- 회귀분석 - 당뇨병 진행도 예측
  - 당뇨병 진행도 예측용 데이터는 442명의 당뇨병 환자를 대상으로한 검사 결과를 나타내는 데이터이다.
  - 10 종류의 독립변수를 가지고 있다. 독립변수의 값들은 모두 스케일링(scaling)되었다.

보스턴 주택 가격 데이터 독립변수	종속변수
age: 나이	1년 뒤 측정한 당뇨병의 진행률
sex: 성별	
bmi: BMI(Body mass index)지수	
bp: 평균혈압	
s1~s6: 6종류의 혈액검사수치	

선형 회귀

- 회귀분석 - 당뇨병 진행도 예측

```
from sklearn.datasets import load_diabetes
import matplotlib.pyplot as plt
import seaborn as sns

diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df["target"] = diabetes.target
df.tail()

sns.pairplot(df[["target", "bmi", "bp", "s1"]])
plt.show()

#독립변수인 BMI지수와 평균혈압이 종속변수인 당뇨병 진행도와 양의 상관관계를 가지는 것을 볼 수 있다.
#또한 두 독립변수 BMI지수와 평균혈압도 서로 양의 상관관계를 가진다.
#이렇게 독립변수끼리 상관관계를 가지는 것을 다중공선성(multicolinearity)이라고 한다.
#다중공선성은 회귀분석의 결과에 영향을 미칠 수 있다
```

선형 회귀

- 회귀분석 – 가상 데이터 예측

$$y = w^T x + b + \epsilon$$

X, y, w = make_regression(n_samples, n_features, bias, noise, random_state, coef=True)
n_samples : 정수 (옵션, 디폴트 100) 표본 데이터의 갯수 N
n_features : 정수 (옵션, 디폴트 100) 독립변수(feature)의 수(차원) M
bias : 실수 (옵션, 디폴트 0) y 절편
noise : 실수 (옵션, 디폴트 0) 출력 즉, 종속변수에 더해지는 잡음 ϵ 의 표준편차
random_state : 정수 (옵션, 디폴트 None)난수 발생용 시드값
coef : 불리언 (옵션, 디폴트 False) True 이면 선형 모형의 계수도 출력
X : [n_samples, n_features] 형상의 2차원 배열, 독립변수의 표본 데이터 행렬 X
y : [n_samples] 형상의 1차원 배열, 종속변수의 표본 데이터 벡터 y
w : [n_features] 형상의 1차원 배열 또는 [n_features, n_targets] 형상의 2차원 배열 (옵션) 선형 모형의 계수 벡터 w , 입력 인수 coef가 True 인 경우에만 출력됨
n_informative : 정수 (옵션, 디폴트 10) , 독립변수(feature) 중 실제로 종속변수와 상관 관계가 있는 독립변수의 수(차원)
effective_rank: 정수 또는 None (옵션, 디폴트 None) 독립변수(feature) 중 서로 독립인 독립변수의 수. 만약 None이면 모두 독립
tail_strength : 0부터 1사이의 실수 (옵션, 디폴트 0.5) effective_rank가 None이 아닌 경우 독립변수간의 상관관계를 결정하는 변수. 0.5면 독립변수간의 상관관계가 없다.

선형 회귀

- 회귀분석 – 가상 데이터 예측

```
from sklearn.datasets import make_regression

X, y, w = make_regression(  n_samples=50, n_features=1, bias=100, noise=10, coef=True, random_state=0 )
xx = np.linspace(-3, 3, 100)
y0 = w * xx + 100
plt.plot(xx, y0, "r-")
plt.scatter(X, y, s=100)
plt.xlabel("x")
plt.ylabel("y")
plt.title("make_regression 예제")
plt.show()
```

```
import matplotlib as mpl
X, y, w = make_regression(     n_samples=300, n_features=2, noise=10, coef=True, random_state=0)

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap=mpl.cm.bone)
plt.xlabel("x1")
plt.ylabel("x2")
plt.axis("equal")
plt.title("두 독립변수가 서로 독립이고 둘 다 종속변수와 상관 관계가 있는 경우")
plt.show()
```

선형 회귀

- 회귀분석 – 가상 데이터 예측

```
X, y, w = make_regression(  n_samples=300, n_features=2, n_informative=1, noise=0, coef=True, random_state=0 )

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap=mpl.cm.bone)
plt.xlabel("x1")
plt.ylabel("x2")
plt.axis("equal")
plt.title("두 독립변수가 서로 독립이고 둘 중 하나만 종속변수와 상관 관계가 있는 경우")
plt.show()
```

```
X, y, w = make_regression( n_samples=300, n_features=2, effective_rank=1, noise=0, coef=True, random_state=0,
tail_strength=0 )

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap=mpl.cm.bone)
plt.xlabel("x1")
plt.ylabel("x2")
plt.axis("equal")
plt.title("두 독립변수가 독립이 아닌 경우")  #다중공선성 문제가 있는 경우
plt.show()
```

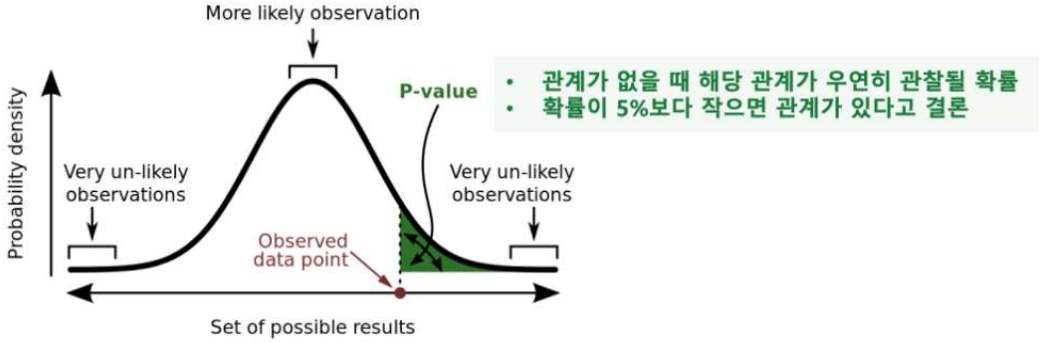
선형 회귀

➢ 선형회귀 (Linear Regression) 분석

- 변수값(매출, 만족도 등)의 차이가 어디에서 비롯되는지 알고자 할 때 사용하는 가장 오래되고 널리 쓰이는 이해하기 쉬운 알고리즘
- 독립변수(X)를 가지고 숫자형 종속변수(Y)를 가장 잘 설명·예측(Best Fit)하는 선형 관계(Linear Relationship)를 찾는 방법
- X와 Y 사이에 선형적 관계가 있다는 가정 하에 실제 Y값(점들)과 예측한 Y값(직선)의 차이를 최소화하는 방정식을 계산

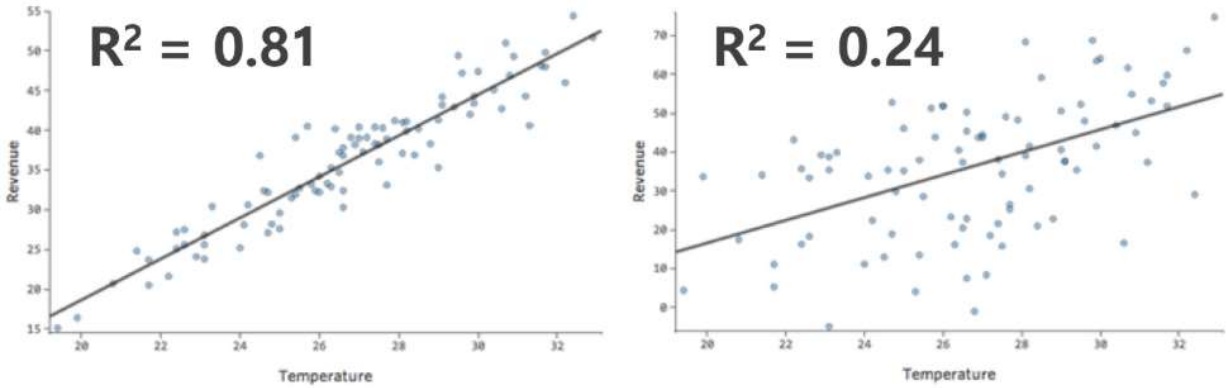
$$Y = b0 + b1X + error$$

- b0 : Y축 절편(Intercept); 예측변수가 0일 때 기대 점수를 나타냄
- b1 : 기울기로 X가 한 단위 증가했을 때의 Y의 평균적 변화값을 나타냄
- P-Value (Probability-Values) : Statistical Significance(통계적 유의성)을 나타내는 수치로 X와 Y 사이에 발견된 관계가 통계적으로 유의미한지 여부를 알려줌
- 데이터를 통해 확인한 관계가 우연히 나왔을 확률
- 예) P값이 0.03이라면 X와 Y 사이에 (선형적) 관계가 없는데도 불구, 데이터 샘플링의 실수로 관계가 우연히 발생했을 확률이 3%
- P-Value값의 절대적 기준은 없고 통상 0.01~0.05 보다 낮다면 유의미하다고 봄



선형 회귀

- 선형회귀 (Linear Regression) 분석
  - R2 (R-SQUARED; 결정계수) : X가 Y를 얼마나 잘 설명/예측하는가를 알려주는 통계량
  - Goodness of Fit: X로 설명할 수 있는 Y 변화량의 크기를 나타내며 0에서 1사의 값을 가짐 (1이면 차이를 100% 설명한다는 의미)



  - y값을 정확히 예측하기 위해선 R^2 값이 중요하지만, 경향성 정보가 중요한 경우 R^2 가 낮다고 꼭 나쁜 모양은 아님

선형 회귀

➢ 선형회귀 (Linear Regression) 분석

- 최소자승법(OLS: Ordinary Least Squares)는 잔차제곱합(RSS: Residual Sum of Squares)를 최소화하는 가중치 벡터를 구하는 방법이다.
- 잔차의 크기(잔차 제곱합)를 가장 작게 하는 가중치 벡터를 구하기 위해 잔차 제곱합을 미분하여 그레디언트(gradient) 벡터를 구합니다.
- 잔차가 최소가 되는 최적화 조건은 그레디언트 벡터가 0벡터이어야 합니다.
- 그레디언트가 0벡터가 되는 관계를 나타내는 식을 직교 방정식(normal equation)이라고 한다.
- 직교 방정식의 특성은 모형에 상수항이 있는 경우에 잔차 벡터의 원소의 합은 0이다. 즉, 잔차의 평균은 0이다.

```
# numPy의 선형대수 기능을 사용하여 OLS 방법으로 선형 회귀분석 수행 코드
from sklearn.datasets import make_regression
import numpy as np
import statsmodels.api as sm

bias = 100
X0, y, w = make_regression(    n_samples=200, n_features=1, bias=bias, noise=10, coef=True, random_state=1)
#statsmodels 패키지의 상수항 결합을 위한 add_constant 함수
X = sm.add_constant(X0)
y = y.reshape(len(y), 1)
w                          #x와 y 관계 y=100+86.44794301x+ϵ
#OLS 해를 직접 이용하는 방법으로 선형 회귀 계수 추정
w = np.linalg.inv(X.T @ X) @ X.T @ y
w                          #최소자승법으로 구한 선형회귀모형  y^=99.79150869+86.96171201x
```

선형 회귀

- 선형회귀 (Linear Regression) 분석

```
#선형 회귀를 통해 구한 가중치 벡터는 정답과 비슷하지만 똑같지는 않다
# 원래 데이터와 비교
x_new = np.linspace(np.min(X0), np.max(X0), 10)
X_new = sm.add_constant(x_new)  # 상수항 결합
y_new = np.dot(X_new, w)

plt.scatter(X0, y, label="원래 데이터")
plt.plot(x_new, y_new, 'rs-', label="회귀분석 예측")
plt.xlabel("x")
plt.ylabel("y")
plt.title("선형 회귀분석의 예")
plt.legend()
plt.show()
```

선형 회귀

- 선형회귀 (Linear Regression) 분석
  - scikit-learn 패키지를 사용한 선형 회귀분석 - linear_model 서브 패키지의 LinearRegression 클래스를 사용

```
from sklearn.linear_model import LinearRegression
#LinearRegression(fit_intercept=True)  fit_intercept 인수는 모형에 상수항이 있는가 없는가를 결정
model = LinearRegression().fit(X0, y)       # #가중치 값을 추정 (상수항 결합을 자동 수행됨)
#coef_ : 추정된 가중치 벡터   ,   intercept_ : 추정된 상수항
print(model.intercept_, model.coef_)
#새로운 입력 데이터에 대한 출력 데이터 예측
model.predict([[-2], [-1], [0], [1], [2]])
```

선형 회귀

➢ 선형회귀 (Linear Regression) 분석

- statsmodels 패키지 OLS 클래스를 사용한 선형 회귀분석
- 1. 독립변수와 종속변수가 모두 포함된 데이터프레임 생성. 상수항 결함은 하지 않아도 된다.
- from_formula 메서드의 인수로 종속변수와 독립변수를 지정하는 formula 문자열을 넣는다.
- data 인수로는 독립변수와 종속변수가 모두 포함된 데이터프레임을 넣는다.
- 2. fit 메서드로 모형 추정. 결과는 별도의 RegressionResults 클래스 객체로 출력된다.
- RegressionResults 클래스 객체는 결과 리포트용 summary 메서드와 예측을 위한 prediction 메서드를 제공한다.
- RegressionResults 클래스는 분석 결과를 다양한 속성에 저장해주므로 추후 사용자가 선택하여 활용할 수 있다.
- params: 가중치 벡터
- resid: 잔차 벡터

```
from sklearn.datasets import load_boston

boston = load_boston()

dfX0 = pd.DataFrame(boston.data, columns=boston.feature_names)
dfX = sm.add_constant(dfX0)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])

model_boston2 = sm.OLS(dfy, dfX)
result_boston2 = model_boston2.fit()
print(result_boston2.summary())
```

선형 회귀

➢ 직선 학습

- 사이킷런의 LinearRegression(선형 회귀)는 특성과 타깃 벡터 사이의 관계가 거의 선형이라고 가정합니다.
- 타깃 벡터에 대한 특성의 효과(계수, 가중치, 파라미터)는 상수입니다.
- y는 타깃이고, xi는 하나의 특성 데이터입니다.
- B1과 B2는 모델을 훈련하여 찾아야 하는 계수입니다.
- e는 오차입니다.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \epsilon$$

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston

boston = load_boston()   # 데이터를 로드하고 두 개의 특성만 선택
features = boston.data[:,0:2]
target = boston.target
regression = LinearRegression()  # 선형 회귀 모델을 만듭니다.
model = regression.fit(features, target)  # 선형 회귀 모델을 훈련합니다.

model.intercept_     # 편향을 확인합니다.
model.coef_          # 특성의 계수를 확인합니다.
target[0]*1000       # 타깃 벡터의 첫 번째 값에 1000을 곱합니다.
```

선형 회귀

- 직선 학습

```
# 첫번째 특성은 인당 범죄율로서 이 특성의 모델 계수는 -0.35입니다.
# 타깃 벡터가 천 달러 단위의 주택 가격이므로 계수에 1,000을 곱하면 인구당 범죄율이 1만큼 증가될 때 주책 가격의 변화를 알 수 있습니다.

model.predict(features)[0]*1000     # 첫 번째 샘플의 타깃 값을 예측하고 1000을 곱합니다.
model.coef_[0]*1000     # 첫 번째 계수에 1000을 곱합니다.
#인구당 범죄율이 1씩 증가될 때마다 주택 가격은 $350 정도 감소한다
```

선형 회귀

- 다항 회귀와 과(대)적합/과소적합
  - 다항(Polynomial) 회귀 : 독립변수의 단항식이 아닌 2차, 3차 방정식과 같은 다항식으로 표현되는 것
  - 다항(Polynomial) 회귀 는 선형 회귀입니다.
  - X에 대해 Target Y값의 관계를 단순 선형 히귀 직선형으로 표현한 것보다 다항 회귀 곡선형으로 표현한 것이 더 예측 성능이 높습니다.
  - 사이킷런은 다항 회귀를 위한 클래스를 명시적으로 제공하지 않습니다.
  - 사이킷런은 다항 회귀 역시 선형 회귀이기 때문에 비선형 함수를 선형 모델에 적용시키는 방법을 사용해 구현합니다.
  - 사이킷런은 PolynomialFeatures 클래스를 통해 피처를 Polynomial(다항식) 피처로 변환합니다.
  - PolynomialFeatures 클래스는 degree 파라미터를 통해 입력 받은 단항식 피처를 degree에 해당하는 다항식 피처로 변환합니다.

  - 다항식의 차수가 높아질수로고 매우 복잡한 피처 간의 관계까지 모델링이 가능합니다.
  - 다항 회귀의 차수(degree)를 높일수록 학습 데이터에만 너무 맞춘 학습이 이뤄져서 정작 테스트 데이터 환경에서는 오히려 예측 정확도가 떨어집니다.

  - 좋은 예측 모델은 학습 데이터의 패턴을 잘 반영하면서도 복잡하지 않은 균형 잡힌(Balanced) 모델을 의미합니다.

선형 회귀

➢ 교차 특성 처리

- 타킷 변수에 영향을 미치면서 다른 특성에 의존하는 특성이 있습니다.
- 사이킷런의 PolynomialFeatures는 교차항을 만들어 의존성을 찾아줍니다.
- 타깃 변수에 대한 특성의 영향이 부분적으로 또 다른 특성에 의존합니다.
- 두 특성값의 곱을 포함하는 새로운 특성을 포함시켜 상호 작용을 나타낼 수 있습니다.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_1 x_2 + \epsilon$$

두 특성의 상호 작용을 나타냅니다

- PolynomialFeatures를 사용해 특성의 모든 조합에 대한 교차항을 만든 다음 모델 선택 전략을 사용해 최선의 모델을 만드는 특성 조합과 교차항을 찾습니다.
- interaction_only=True를 지정하면 PolynomialFeatures가 교차항만 반환합니다.
- 기본적으로 PolynomialFeatures를 절편이라고 부르는 1로 채워진 특성을 추가합니다.
- include_bias=False는 절편 1로 채워진 특성을 추가하지 않습니다.

선형 회귀

➢ 교차 특성 처리

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures

boston = load_boston()                    # 데이터를 로드하고 두 개의 특성만 선택
features = boston.data[:,0:2]
target = boston.target

# 교차 항을 만듭니다.
interaction = PolynomialFeatures(     degree=3, include_bias=False, interaction_only=True)
features_interaction = interaction.fit_transform(features)

regression = LinearRegression()  # 선형 회귀 모델 객체 생성
model = regression.fit(features_interaction, target)  # 선형 회귀 모델 훈련
features[0]   # 첫 번째 샘플의 특성 값을 확인

import numpy as np
# 각 샘플에서 첫 번째와 두 번째 특성을 곱합니다.
interaction_term = np.multiply(features[:, 0], features[:, 1])
interaction_term[0]                # 첫 번째 샘플의 교차 항을 확인.
features_interaction[0]            # 첫 번째 샘플의 값을 확인
```

선형 회귀

- 비선형 관계 학습
  - 선형 회귀 모델에 다항 특성을 추가하여 다항 회귀를 만듭니다.
  - 다항 회귀는 선형 회귀의 확장하여 비선형 관계를 모델링합니다.
  - 다항 회귀는 다항 특성을 추가하여 이를 다항 함수로 변환합니다.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_1^2 + \ldots + \hat{\beta}_d x_1^d + \epsilon$$

선형 회귀

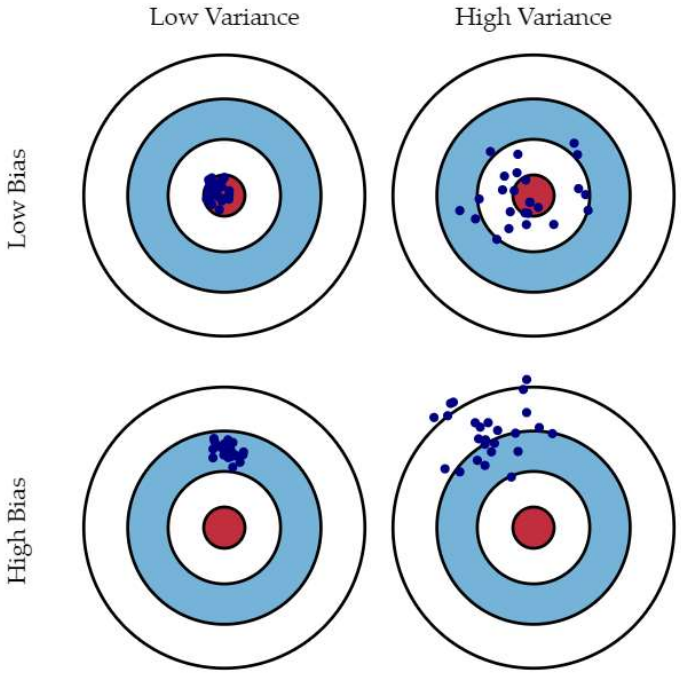
➢ 비선형 관계 학습

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures

boston = load_boston()                    # 데이터를 로드하고 하나의 특성을 선택
features = boston.data[:,0:1]
target = boston.target

# 다항 특성 x^2와 x^3를 만듭니다.
polynomial = PolynomialFeatures(degree=3, include_bias=False)
features_polynomial = polynomial.fit_transform(features)

regression = LinearRegression()  # 선형 회귀 모델 객체 생성
model = regression.fit(features_polynomial, target)   # 선형 회귀 모델 훈련

features[0]                       # 첫 번째 샘플을 확인
features[0]**2                    # 첫 번째 샘플을 x^2로 거듭제곱합니다.
features[0]**3                    # 첫 번째 샘플을 x^2로 세제곱합니다.
features_polynomial[0]            # 첫 번째 샘플의 x, x^2,x^3 값을 확인
```

선형 회귀

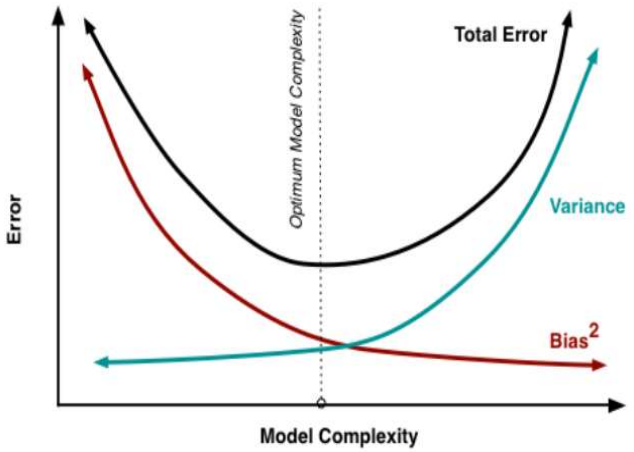
- 실제값과 예측값의 차이를 최소화하는 것에만 초점을 맞춘 단순 선형 회귀는 학습 데이터에 과적합되는 문제를 수반할 가능성이 높습니다.
- 과적합 문제를 해결하기 위해 규제(Regularization)를 선형 회귀에 도입했습니다.

- Degree 1과 같은 모델은 매우 단순화된 모델로서 지나치게 한 방향성으로 치우친 경향이 있습니다. (고편향 Hight Bias성을 가졌다고 표현합니다.)
- Degree 15과 같은 모델은 학습 데이터 하나 하나의 특성을 반영하면서 매우 복잡한 모델이 되었고 지나치게 높은 변동성을 가지게 되었습니다. (고분산 High Variance성을 가졌다고 표현합니다.)
- 저편향/저분산(Low Bias/Low Vaiance)은 예측 결과가 실제 결과에 매우 잘 근접하면서도 예측 변동이 크지 않고 특정 부분에 집중돼 있는 아주 뛰어난 성능을 보여줍니다.
- 저편향/고분산(Low Bias/High Vaiance)은 예측 결과가 실제 결과에 비교적 근접하지만, 예측 결과가 실제 결과를 중심으로 꽤 넓은 부분에 분포돼 있습니다.
- 고편향/저분산(High Bias/Low Vaiance)은 정확한 결과에서 벗어나면서도 예측이 특정 부분에 집중돼 있습니다.
- 고편향/고분산(High Bias/High Vaiance)은 정확한 결과에서 벗어나면서도 예측이 넓은 부분에 집중돼 있습니다.



선형 회귀

- 편향과 분산은 한 쪽이 높으면 한 쪽이 낮아지는 경향이 있습니다.
- 편향이 높으면 분산이 낮아지고(과소적합) 반대로 분산이 높으면 편향이 낮아집니다.(과적합)
- 편향과 분산의 관계에 따른 전체 오류 값(Total Error)의 변화를 잘 보여줍니다.
- 편향이 너무 높으면 전체 오류가 높습니다.
- 편향을 점점 낮추면 동시에 분산이 높아지고 전체 오류도 낮아지게 됩니다.
- 편향을 낮추고 분산을 높이면서 전체 오류가 가장 낮아지는 '골디락스' 지점을 통과하면서 분산을 지속적으로 높이면 전체 오류 값이 오히려 증가하면서 예측 성능이 다시 저하됩니다.
- 높은 편향/낮은 분산에서 과소적합되기 쉬우며 낮은 편향/높은 분산에 과적합되기 쉽습니다.
- 편향과 분산이 서로 트레이드오프를 이루면서 오류 cost 값이 최대로 낮아지는 모델을 구축하는 것이 가장 효율적인 머신러닝 예측 모델을 만드는 방법



선형 회귀

- 비용함수는 학습 데이터의 잔차 오류 값을 최소로 하는 RSS 최소화 방법과 과적합을 방지하기 위해 회귀 계수 값이 커지지 않도록 하는 방법이 서로 균형을 이뤄야 합니다.
- alpha는 학습 데이터 적합 정도와 회귀 계수 값의 크기 제어를 수행하는 튜닝 파라미터입니다.

$$\text{비용 함수 목표} = Min(RSS(W) + alpha * ||W||_2^2)$$

- alpha를 0에서부터 지속적으로 값을 증가시키면 회귀 계수 값의 크기를 감소시킬 수 있다. 이처럼 비용 함수에 alpha 값으로 페널티를 부여해 회귀 계수 값의크기를 감소시켜 과적합을 개선하는 방식을 규제(Regularization)라고 부른다.
- alpha = 0인 경우는 W가 커도 $alpha * ||W||_2^2$ 가 0이 되어 비용 함수는 Min(RSS(W))
- alpha = 무한대인 경우 $alpha * ||W||_2^2$ 도 무한대가 되므로 비용 함수는 W를 0에 가깝게 최소화 해야 함
- L2 규제는 $alpha * ||W||_2^2$ 같이 W의 제곱에 대해 패널티를 부여하는 방식을 말한다.
- L2 규제를 적용한 회귀를 릿지(Ridge) 회귀
- 라쏘(Lasso) 회귀는 L1 규제를 적용한 회귀
- L1 규제는 $alpha * ||W||_2^2$ W의 절대값에 대해 패널티를 부여한다.
- L1 규제를 적용하면 영향력이 크지 않은 회귀 계수 값을 0으로 변환한다.

선형 회귀

➢ 규제로 분산 축소

- 정규화는 정규화항을 통해 모델에 미치는 차원의 수의 수를 감소시키기 때문에 overfitting을 방지하게 됩니다.
- 일반적인 회귀방법에서 비용함수는 MSE를 최소화하는 방향으로 나아가게 됩니다. 일반적인 회귀방법에서 데이터의 특징수가 많아질수록(차원이 증가할수록) overfitting에 대한 위험성이 커지게 됩니다.
- 이를 막기위해 정규화 항을 사용하게 되는데요. MSE + regular-term으로 비용함수를 재정의하게 됩니다.
- 비용함수를 최소화하는 방향에선 regular-term또한 최소화가 되어야 할겁니다.
- 최소화를 진행하게 되면서 가중치가 낮은 항은 정규화 방법에 따라 0으로 수렴하여 사용하지 않게 되거나 0에 가까운 수가 되어 모델에 미치는 영향이 덜해지게 됩니다.

- 릿지 회귀 : L2-Norm을 사용한 회귀입니다. 이 회귀방법은 일반적으로 영향을 거의 미치지 않는 특성에 대하여 0에 가까운 가중치를 주게 됩니다.

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

- 라쏘 회귀 : L1-Norm을 사용한 회귀입니다. 특성값의 계수가 매우 낮다면 0으로 수렴하게 하여 특성을 지워버립니다. 특성이 모델에 미치는 영향을 0으로 만든다는 것은 bias를 증가 시켜 overfitting을 방지한다는 의미가 됩니다.

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} | \theta_i |$$

- 엘라스틱 넷 : 라쏘회귀와 릿지회귀의 최적화 지점이 서로 다르기 때문에 두 정규화 항을 합쳐서 r로 규제정도를 조절하여 준다.

$$J(\theta) = \mathrm{MSE}(\theta) + r\alpha \sum_{i=1}^{n} | \theta_i | + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$

선형 회귀

➢ 규제로 분산 축소

- 리지 회귀나 라소 회귀와 같이 축소 페널티가 포함된 학습 알고리즘을 사용합니다.
- 선형 회귀에서는 모델이 정답(Yi)과 예측(yi) 사이의 제곱 오차 합 또는 잔차 제곱합(RSS)을 최소화하기 위해 훈련합니다

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

```
from sklearn.linear_model import Ridge
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler

boston = load_boston()                          # 데이터 로드
features = boston.data
target = boston.target

scaler = StandardScaler()                       # 특성을 표준화
features_standardized = scaler.fit_transform(features)
regression = Ridge(alpha=0.5)                   # alpha 값을 지정한 릿지 회귀를 만듭니다.
model = regression.fit(features_standardized, target)          # 선형 회귀 모델을 훈련합니다.
```

선형 회귀

➢ 규제로 분산 축소

- RSS와 전체 계숫값의 합인 페널티를 최소화합니다.
- 모델을 축소시키려는 경향이 있기 때문에 페널티를 축소 페널티라고 부릅니다.
- 규제를 적용한 선형 회귀 - 리지 회귀, 라소 회귀
- 리지 회귀에서 축소 페널티는 모든 계수의 제곱합에 튜닝 파라미터를 곱한 것입니다.

$$\mathrm{RSS} + \alpha \sum_{j=1}^{p} \hat{\beta}_j^2$$

```
from sklearn.linear_model import RidgeCV

regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])                  # 세 개의 alpha 값에 대한 릿지 회귀 객체 생성
model_cv = regr_cv.fit(features_standardized, target)       # 선형 회귀 모델  훈련
model_cv.coef_                                              # 계수 확인
model_cv.alpha_                                             # alpha 값을 확인

regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)            # 5-폴드 교차검증을 사용하여 릿지 회귀 객체 생성
model_cv = regr_cv.fit(features_standardized, target)       # 선형 회귀 모델을 훈련합니다.
model_cv.alpha_                                             # alpha 값을 확인

regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)            # 5-폴드 교차검증을 사용하여 릿지 회귀 객체 생성

model_cv = regr_cv.fit(features_standardized, target)       # 선형 회귀 모델을 훈련
model_cv.alpha_                                             # alpha 값을 확인
```

선형 회귀

- 규제로 분산 축소
  - 라소 회귀는 축소 페널티가 모든 계수의 절댓값 합에 튜닝 하이퍼라미터를 곱한 것입니다.
  - 리지 회귀가 라소보다 조금 더 좋은 예측을 만듭니다.
  - 라소 회귀는 더 이해하기 쉬운 모델을 만듭니다.
  - 리지와 라소 페널티 사이에 균형을 맞추고 싶다면 엘라스틱 넷을 사용할 수 있습니다.
  - 리지와 라소 회귀는 최소화하려는 손실 함수에 계숫값을 포함시킴으로써 크고 복잡한 모델을 만듭니다.
  - 하이퍼파라미터 a는 계수를 얼마나 불리하게 만들지 조절합니다.

$$\frac{1}{2n}\mathrm{RSS}+\alpha\sum_{j=1}^{p}\left|\hat{\beta}_j\right|$$

  - a값이 클수록 더 간단한 모델을 만듭니다.
  - 이상적인 a 값을 구하려면 다른 하이퍼라미터와 같이 튜닝해야만 합니다.
  - a는 alpha 매개변수를 사용해 지정합니다.
  - 사이킷런의 RidgeCV 클래스를 사용하면 좋은 a값을 선택할 수 있습니다.
  - RidgeCV 클래스의 cv 매개변수를 사용해 교차검증 방식을 지정할 수 있습니다.
  - 기본값은 None으로 LOOCV 방식을 사용합니다.
  - 정수를 지정하면 GridSearchCV를 사용하여 교차검증을 수행합니다.

선형 회귀

- 라쏘 회귀
  - W의 절대값에 페널티를 부여하는 L1 규제를 선형 회귀에 적용한 것
  - RSS와 전체 계숫값의 합인 페널티를 최소화합니다.

$$RSS(W) + alpha1 * ||W||_1$$

  - L2 규제가 회귀 계수의 크기를 감소시키는 데 반해 L2 규제는 불필요한 회귀 계수를 급격하게 감소시켜 0으로 만들고 제거 합니다.
  - L1 규제는 적절한 피처만 회귀에 포함시키는 피처 선택의 특성을 가지고 있습니다.

선형 회귀

➢ 라쏘 회귀로 특성 축소

- 라소 회귀를 사용하여 특성 수를 줄여 선형 회귀 모델을 단순화 할 수 있습니다
- 모델의 계수를 0까지 축소시킬 수 있습니다.
- RSS와 전체 계숫값의 합인 페널티를 최소화합니다.

```
from sklearn.linear_model import Lasso
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler

boston = load_boston()                                  # 데이터 로드
features = boston.data
target = boston.target
scaler = StandardScaler()                               # 특성 표준화
features_standardized = scaler.fit_transform(features)

regression = Lasso(alpha=0.5)                           # alpha 값을 지정한 라쏘 회귀 객체 생성
model = regression.fit(features_standardized, target)   # 선형 회귀 모델 훈련
model.coef_                                             # 계수 확인

regression_a10 = Lasso(alpha=10)                        # 큰 alpha 값을 지정한 라쏘 회귀 객체 생성
model_a10 = regression_a10.fit(features_standardized, target)
model_a10.coef_
```

선형 회귀

➢ 라쏘 회귀로 특성 축소

- alpha값이 너무 크게 증가하면 어떤 특성도 사용되지 않습니다.
- 라소의 alpha 값을 찾기 위해 LassoCV 클래스를 사용할 수 있습니다. (cv 매개변수 기본값은 3으로 3-폴드 교차검증을 사용합니다.)
- LassoCV는 alphas 매개변수에 탐색할 값을 명시적으로 지정하지 않고 n_alphs 매개변수를 사용해 자동으로 탐색 대상 값을 생성할 수 있습니다.

```
from sklearn.linear_model import LassoCV

# 세 개의 alpha 값에 대한 라쏘 회귀를 만듭니다.
lasso_cv = LassoCV(alphas=[0.1, 1.0, 10.0], cv=5)

model_cv = lasso_cv.fit(features_standardized, target)      # 선형 회귀 모델 훈련
model_cv.coef_                                              # 계수를 확인
model_cv.alpha_                                             # alpha 값을 확인

# 1000개의 alpha 값을 탐색하는 라쏘 회귀를 만듭니다.
lasso_cv = LassoCV(n_alphas=1000, cv=5)

model_cv = lasso_cv.fit(features_standardized, target)      # 선형 회귀 모델 훈련
model_cv.alpha_                                              # 계수를 확인
lasso_cv.alphas_                                            # alpha 값을 확인
```

선형 회귀

- Elastic Net 회귀
  - L2 규제와 L1 규제를 결합한 회귀
  - 비용함수

$$RSS(W) + alpha2 * ||W||_2^2 + alpha1 * ||W||_1$$

  - 비용함수를 최소화하는 W를 찾는 것
  - 라쏘 회귀가 서로 상관관계가 높은 피처들의 경우에 이들 중에서 중요 피처만을 셀렉션하고 다른 피처들은 모두 회귀 계수를 0으로 만드는 성향이 강합니다.
  - ElasticNet 클래스

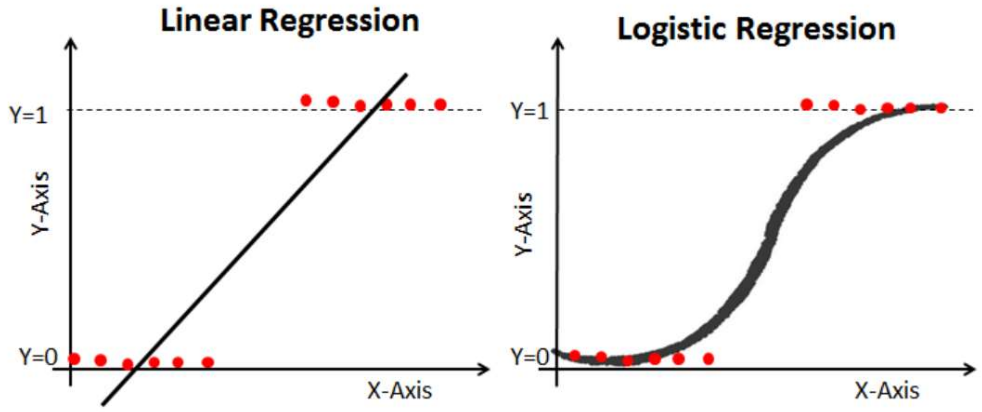
선형 회귀

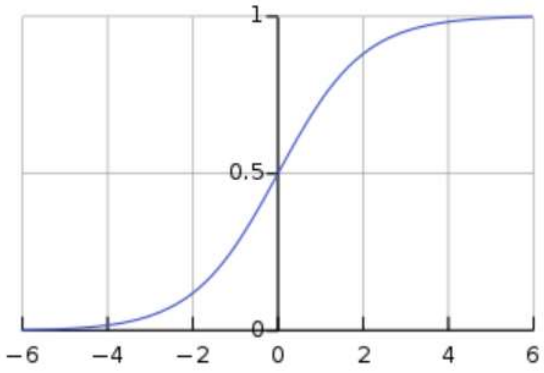
- 선형 회귀 모델을 위한 데이터 변환
  - 선형 모델을 기반으로 하는 선형 회귀는 데이터 값의 분포도와 인코딩 방법에 많은 영향을 받을 수 있습니다.
  - 선형 회귀는 데이터 값의 분포도가 정규 분포와 같이 종 모양의 형태를 선호하며,
  - 특히, 타깃값의 분포도가 왜곡(Skew)되지 않고 정규 분포 형태로 되어야 예측 성능을 저하시키지 않습니다.
  - 타깃값의 경우 정규 분포 형태가 아니라 특정값의 분포가 치우친 왜곡(Skew)된 형태의 분포도일 경우 예측 성능에 부정적인 영향을 미칠 가능성이 높습니다.
  - 피처값 역시 결정값보다는 덜하지만 왜곡된 분포도로 인해 예측 성능에 부정적인 영향을 미칠 수 있습니다.
  - 선형 회귀 모델을 적용하기 전에 먼저 데이터에 대한 스케일링/정규화 작업을 수행하는 것이 일반적입니다.

선형 회귀

- 로지스틱 회귀
  - 선형 회귀 방식을 분류에 적용한 알고리즘
  - 회귀가 선형인가 비선형인가는 독립변수가 아닌 가중치(weight)변수가 선형인지 아닌지를 따릅니다.
  - 시그모이드 함수 최적선을 찾고 시그모이드 함수의 반환 값을 확률로 간주해 확률에 따라 분류를 결정한다

  - 독립변수(피처)를 시그모이드 함수에 입력해 반환된 결과를 확률값으로 변환해 예측 레이블을 결정합니다.





  - 시그모이드 함수는 x 값이 +, -로 아무리 커지거나 작아져도 y 값은 항상 0과 1 사이 값을 반환한다. x 값이 커지면 1에 근사하며 x 값이 작아지면 0에 근사한다. 그리고 x가 0일 때는 0.5이다.

$$y = \frac{1}{1+e^{-}x}$$

선형 회귀

- 로지스틱 회귀
  - 사이킷런 LogisticRegression 클래스의 주요 하이퍼 파라미터로 penalty와 C가 있다.
  - penalty는 규제의 유형을 설정하며 'l2'로 설정 시 L2 규제를, 'l1'으로 설정 시 L1 규제를 뜻한다. 기본은 'l2'이다.
  - C는 규제 강도를 조절하는 alpha 값의 역수이다.

$$C = \frac{1}{alpha}$$

  - 로지스틱 회귀는 희소한 데이터 세트 분류에도 뛰어난 성능을 보여서 텍스트 분류에서도 자주 사용됩니다.