



5장. 회귀



여러 개의 독립 변수와 한 개의 종속 변수 간의 상관관계를 모델링 하는 기법

1 회귀 소개

회귀 분석은 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법

: 주어진 피처와 결정 값 데이터 기반에서 학습을 통해 최적의 회귀계수(Regression coefficients)를 찾아내는 것

- 회귀계수: 독립변수 값에 영향을 미치는 값

독립변수 갯수	회귀 계수의 결합
1개 : 단일 회귀	선형 : 선형 회귀
1개(n) : 다항 회귀	선형 : 선형 회귀
여러개 : 다중 회귀	비선형 : 비선형 회귀

선형 회귀 : 실제 값과 예측 값의 차이(오류의 제곱 값)를 최소화하는 직선형 회귀선을 최적화하는 방식

- 일반 선형 회귀 : 실제-예측값의 RSS(Residual 잔여의 Sum of Squares)를 최소화할 수 있도록 회귀 계수를 최적화하며, 규제를 적용하지 않은 모델
- 릿지 : 선형 회귀 + L2 규제 \Rightarrow L2 : 상대적으로 큰 회귀 계수 값의 예측 영향도를 감소시키기 위해 회귀 계수값을 더 작게 만드는 규제
- 라쏘 : 선형 회귀 + L1 규제 \Rightarrow L1 : (L2는 회귀계수값 크기 줄이는데 반해) 예측 영향력이 작은 피처의 회귀 계수를 0으로 만들어, 회귀예측시 피처가 선택되지 않게 하는 것 \rightarrow L1 규제는 피처 선택 기능으로 불림
- 엘라스틱넷 : L2, L1 규제를 함께 결합한 모델 \rightarrow 주로 피처가 많은 데이터 세트에 적용됨
- 로지스틱 회귀 : 분류에 사용되는 선형 회귀 모델

2 단순 선형 회귀를 통한 회귀 이해

: 독립 변수도 하나, 종속 변수도 하나인 선형 회귀

오류 합 계산


- MAE : 절댓값 취한 후 더하기
- RSS : 제곱 후 더하기
→ $Error^2 = RSS$
- 회귀에서 이 RSS 값은 비용(Cost)이며, w변수(회귀계수) 구성되는 RSS를 비용 함수라고 함
 - 회귀 알고리즘은 이 비용 함수가 반환하는 값(즉, 오류 값)을 지속해서 감소시키고 최종적으로는 더 이상 감소하지 않는 최소의 오류값을 구하는 것
 - 비용 함수를 손실 함수(loss function)라고도 함

3 경사 하강법

[참고자료]

경사하강법(gradient descent) - 공돌이의 수학정리노트 (Angelo's Math Notes)

Gradient Descent 방법은 1차 미분계수를 이용해 함수의 최소값을 찾아가는 iterative한 방법이다. Step size를 조정해가며 최소값을 찾아가는 과정을 관찰해보자.gradient descent 방법의 직관적 의미gradient descent 방법은 stee...

 https://angelayeo.github.io/2020/08/16/gradient_descent.html

- 단순 선형 회귀로 이해해보기
 - 모델 : $f(x)=w_0+w_1*x$
 - 실젯값 : $Y_i=w_0+w_1*X_i \cdots Error_i$
 - 예측값 : $Y^{\wedge}=w_0+w_1*X$
 -

- **경사하강법**

: 점진적으로 반복적인 계산을 통해 W를 업데이트하면서 오류값이 최소가 되는 W를 구하는 방식

- 2차함수의 최저점은 미분 값인 1차 함수의 기울기가 가장 최소일 때

$$\begin{aligned} R(w) &= \frac{1}{N} \sum_{i=1}^N (y_i - (w_0 + w_1 * x_i))^2 \\ &= \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \\ &= \text{평균}((\text{실제 값}_i - \text{예측 값}_i)^2) \\ &= \text{평균(오차제곱)} \\ &= MSE \end{aligned}$$

- $R(w_0, w_1)$ 를 각 w_0, w_1 로 편미분

예제) 다음을 편미분 하여라.

$$y = f(x_1, x_2) = 3x_1^4 + 2x_1^2x_2^2 + 7x_2^4$$

풀이)

$$\frac{\partial y}{\partial x_1} = 12x_1^3 + 4x_1x_2^2$$

$$\frac{\partial y}{\partial x_2} = 4x_1^2x_2 + 28x_2^3$$

- $\partial R(w)/\partial w_1 = \frac{2}{N} \sum_{i=1}^N -x_i * (y_i - (w_0 + w_1x_i)) = -\frac{2}{N} \sum_{i=1}^N x_i * (\text{실제 값}_i - \text{예측 값}_i)$
- $\partial R(w)/\partial w_0 = \frac{2}{N} \sum_{i=1}^N -(y_i - (w_0 + w_1x_i)) = -\frac{2}{N} \sum_{i=1}^N (\text{실제 값}_i - \text{예측 값}_i)$

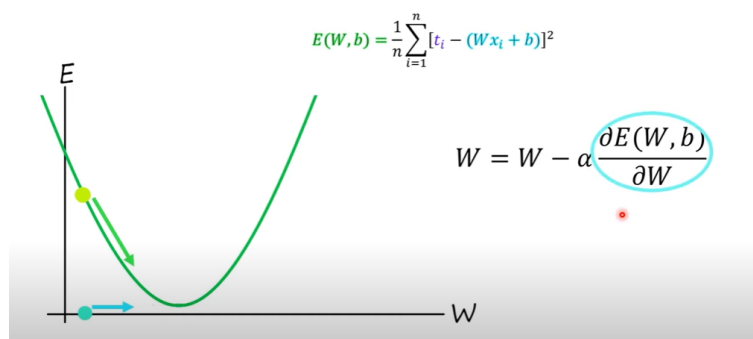
w_1, w_0 의 편미분 결과값인 $-\frac{2}{N} \sum_{i=1}^N x_i * (\text{실제 값}_i - \text{예측 값}_i)$, $-\frac{2}{N} \sum_{i=1}^N (\text{실제 값}_i - \text{예측 값}_i)$ 을 반복적으로 보정하면서, w_1, w_0 값을 업데이트 하면서 비용 함수 $R(w)$ 를 최소가 되는 w_0, w_1 을 구할 수 있음

•

- 이때, 편미분 값이 너무 클 수 있기 때문에 보정계수 η 를 곱하는데, 이를 학습률(learning_rate)이라고 한다.
- 업데이트는 편미분 결과값을 마이너스(-)하면서 적용

- 새로운 $w_1 = \text{이전 } w_1 - (-\eta \frac{2}{N} \sum_{i=1}^N x_i (\text{실제 값}_i - \text{예측 값}_i))$
- 새로운 $w_0 = \text{이전 } w_0 - (-\eta \frac{2}{N} \sum_{i=1}^N (\text{실제 값}_i - \text{예측 값}_i))$

- 이 과정을 반복적으로 적용하면서, 비용함수가 최소가 되는 값을 찾는 것



- 경사 하강법의 일반적인 프로세스

- [Step 1] : w_1, w_0 을 임의의 값으로 설정하고 첫 비용 함수의 값을 계산
- [Step 2] : w_1 을 $w_1 + \eta \frac{2}{N} \sum_{i=1}^N x_i (\text{실제값}_i - \text{예측값}_i)$ 으로, w_0 을 $w_0 + \eta \frac{2}{N} \sum_{i=1}^N (\text{실제값}_i - \text{예측값}_i)$ 으로 업데이트한 후, 다시 비용 함수의 값을 계산
- [Step 3] : 비용 함수의 값이 감소했으면, 다시 [Step 2]를 반복, 더 이상 비용 함수 값이 감소하지 않으면 그때의 w_1, w_0 를 구하고 반복 중지

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

### --- 실제값을 Y=4X+6 시뮬레이션하는 데이터 값 생성 -----

np.random.seed(0)
# y = 4X + 6 식을 근사(w1=4, w0=6). random 값은 Noise를 위해 만들
X = 2 * np.random.rand(100,1)
y = 6 + 4 * X + np.random.randn(100,1)
print(type(X))

### --- w0과 w1의 값을 최소화 할 수 있도록 업데이트 수행하는 함수 생성 -----

# w1 과 w0 를 업데이트 할 w1_update, w0_update를 반환.
def get_weight_updates(w1, w0, X, y, learning_rate=0.01):
    N = len(y)

    # 먼저 w1_update, w0_update를 각각 w1, w0의 shape와 동일한 크기를 가진 0 값으로 초기화
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)

    # 예측 배열 계산하고 예측과 실제 값의 차이 계산
    y_pred = np.dot(X, w1.T) + w0 # 예측값
    diff = y - y_pred # 실제값 - 예측값
    # diff == error

    # w0_update를 dot 행렬 연산으로 구하기 위해 모두 1값을 가진 행렬 생성
    w0_factors = np.ones((N,1))

    # w1과 w0을 업데이트할 w1_update와 w0_update 계산
    w1_update = -(2/N)*learning_rate*(np.dot(X.T, diff)) # w1의 편미분 값
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T, diff)) # w0의 편미분 값

    return w1_update, w0_update

### --- 반복적으로 경사 하강법을 이용하여 get_weight_updates()를 호출하여 w1과 w0를 업데이트 하는 함수 생성 ---

# 입력 인자 iters로 주어진 횟수만큼 반복적으로 w1과 w0를 업데이트 적용함.
def gradient_descent_steps(X, y, iters=10000):
    # w0와 w1을 모두 0으로 초기화.
    w0 = np.zeros((1,1))
    w1 = np.zeros((1,1))

    # 인자로 주어진 iters 만큼 반복적으로 get_weight_updates() 호출하여 w1, w0 업데이트 수행.
    for ind in range(iters):
        w1_update, w0_update = get_weight_updates(w1, w0, X, y, learning_rate=0.01) # w1, w0 편미분값 return
        w1 = w1 - w1_update # 새로운 w1
        w0 = w0 - w0_update # 새로운 w0

    return w1, w0

### --- 예측 오차 비용을 계산할 수행하는 함수 생성 및 경사 하강법 수행 -----

def get_cost(y, y_pred):
    N = len(y)
    cost = np.sum(np.square(y - y_pred))/N # 평균( (실제값 - 예측값)^2 ) = RMSE
    return cost
```

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

- 경사하강법은 모든 학습 데이터에 반복적으로 업데이트 하기에 시간이 매우 오래 걸린다는 단점 ⇒ 확률적 경사하강법을 대신 사용
- (미니배치) 확률적 경사하강법(Stochastic *통계학의* 확률적인 Gradient Descent): 일부 데이터만을 이용해 w가 업데이트되는 값 계산 ⇒ SGD
 - 전반적으로 경사하강법과 비슷하지만, 전체 X, y 데이터에서 랜덤하게 batch_size만큼 데이터를 추출해서 w1, w0을 업데이트 함
- 피처가 여러개인 경우 어떻게 회귀 계수 도출할 수 있을까? (↔ 지금까지는 피처 1개, 독립변수 1개인 단순 선형 회귀의 경사하강법)
 - 피처 1개인 경우 : $Y^{\wedge}=w_0+w_1 \cdot X$ 로
 - 피처가 M개(X_1, X_2, \dots, X_m) 인 경우 : $Y^{\wedge}=w_0+w_1 \cdot X_1+w_2 \cdot X_2+\dots+w_{100} \cdot X_{100}$
 - 회귀 계수도 M+1개 (1개는 w_0)
 - 데이터가 N개이고 피처가 M개인 입력 행렬을 X_{mat} , 회귀 계수 w_1, w_2, \dots, w_{100} 을 W 배열로 표기하면
 $\Rightarrow Y^{\wedge}=np.dot(X_{mat}, W^T)+w_0$
 - w_0 을 W배열에 포함시키기 위해서, X_{mat} 의 맨 처음 열에 모든 데이터 값이 1인 피처 Feat 0을 추가 하면 ⇒

$$Y^{\wedge}=X_{mat} \cdot W^T$$

4 사이킷런 LinearRegression을 이용한 보스턴 주택 가격 예시

4-1. LinearRegression 클래스 - Ordinary Least Squares

LinearRegression 클래스는 RSS를 최소화해 OLS(Ordinary Least Squares) 추정 방식으로 구현됨

- `fit()` 메서드로 X, y 배열 받으면, 회귀 계수(Coefficients)인 W를 `coef_` 속성에 저장
- 입력 파라미터에서, `normalize=True` 로 설정하면 회귀 수행 전, 입력 데이터 세트를 정규화함 *`default=False`
- OLS 기반 회귀 계수 계산은 입력 피처의 독립성에 많은 영향을 받아, 피처 상관관계가 높은 경우 분산이 매우 커져 오류에 매우 민감해짐
 - 다중회귀(비선형) ⇒ 다중 공선성 문제 (multi-collinearity)
 - 그래서 일반적으로 상관 관계가 높은 피처가 많은 경우 독립적인 중요한 피처만 남기고 제거 or 규제 및 PCA로 차원 축소 수행하기도 함

4-2. 회귀 평가 지표

- MAE(Mean Absolute Error)

- $1/N \sum_{i=1}^N |Y_i - \hat{Y}_i|$
- 실제값과 예측값의 차이를 절대값으로 변환해 평균한 것
- 사이킷런 평가지표: `metrics.mean_absolute_error`
- Scoring 함수 적용값 : 'neg_mean_absolute_error'
 - 'neg_' \Rightarrow -1을 곱해서 반환 (scoring함수는 값이 클수록 좋은 평가 결과로 보기 때문)
 - $10 > 1 \Rightarrow -10 < -1 == \text{neg_mean_absolute_error} \Rightarrow -1 * \text{metrics.mean_absolute_error}$

- MSE(Mean Squared Error)

- $MSE = 1/N \sum_{i=1}^N (Y_i - \hat{Y}_i)^2$
- 실제값과 예측값의 차이를 제곱해 평균한 것
- `metrics.mean_squared_error`
- Scoring 함수 적용값 : 'neg_mean_squared_error'

- RMSE(Root MSE)

- $RMSE = \sqrt{1/N \sum_{i=1}^N (Y_i - \hat{Y}_i)^2}$
- MSE는 실제 오류 평균보다 커지므로 보정한 것
- 사이킷런은 RMSE를 제공하지 않음

- R^2

- $R^2 = \frac{\text{예측분산}}{\text{실제분산}}$
- $R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\text{평균})^2} = 1 - \frac{\sum(\text{오차})^2}{\sum(\text{편차})^2}$
- 1에 가까워질 수록 예측 정확도가 좋은 것
 - $R^2 = 1 = (1 - 0) \Rightarrow \frac{\sum(\text{오차})^2}{\sum(\text{편차})^2} = \frac{\sum(y-\hat{y})^2}{\sum(y-\text{평균})^2} \Rightarrow (y - \hat{y}) = 0 \Rightarrow y = \hat{y} \Rightarrow$
예측값이 실제값과 같다
 - $R^2 = 0 = (1 - 1) \Rightarrow \frac{\sum(\text{오차})^2}{\sum(\text{편차})^2} = \frac{\sum(y-\hat{y})^2}{\sum(y-\text{평균})^2} \Rightarrow (y - \hat{y}) = (y - \text{평균}) \Rightarrow$
 $\text{평균} = \hat{y} \Rightarrow$ 예측값이 평균값과 같다 (예측하나마나다)
- `metrics.r2_score`
- Scoring 함수 적용값 : 'r2'

4-3. 선형 회귀 Linear Regression

: 예측값과 실제값의 RSS를 최소화해 OLS(Ordinary Least Squares) 추정방식으로 구현한 클래스

1. 입력 파라미터

파라미터	default	설
------	---------	---

fit_intercept	True	Boolean, intercept(절편)값을 계산할 것인지 결정
normalize	False	Boolean, True면 회귀를 수행하기 전에 입력 데이터 세트 정규화, fit_intercept=False 인 경우 이 파라미터는 무시됨

2. 속성

- `coef_`: `fit()` 메서드를 수행했을 때, 회귀 계수가 배열 형태로 저장되는 속성, `shape = (Target 개수, 피쳐 개수)`
- `intercept_`: 절편 값
- Seaborn의 `regplot()`: x, y축 값의 산점도와 선형 회귀 직선을 그려준다

5 다항 회귀와 과(대)적합/과소적합 이해



: 회귀가 독립변수의 단항식이 아닌 2차, 3차 방정식과 같은 다항식으로 표현되는 것(하지만 선형회귀다!)

5-1. 다항 회귀 이해

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

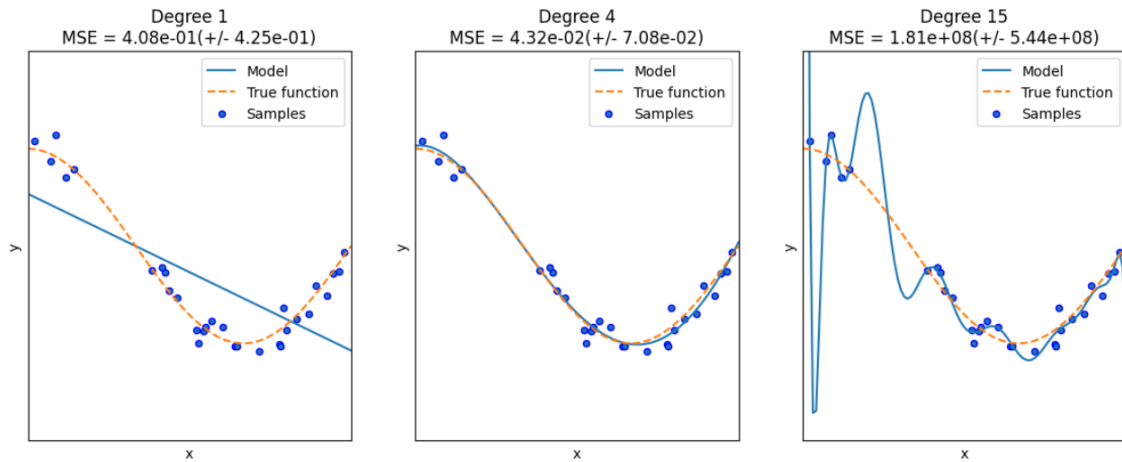
- 선형회귀 / 비선형 회귀를 나누는 기준 : 회귀 계수가 선형/비선형인지에 따른 것(독립변수의 선형/비선형은 X)
- 사이킷런은 다항회귀를 위한 클래스를 명시적으로 제공 X
 - 비선형 함수를 선형 모델에 적용시키는 방법을 사용해 구현
 - `PolynomialFeature()` 로 피쳐를 변환한 후에 `LinearRegression` 클래스로 다항 회귀 구현

5-2. 과소적합 및 과적합

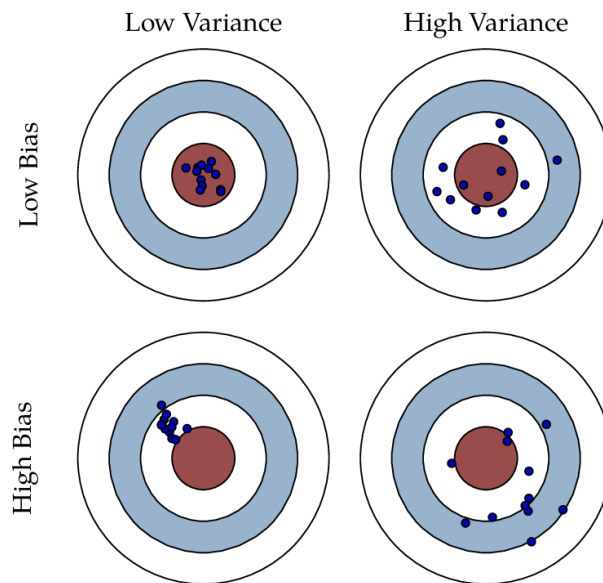
- 차수가 높아질 수록 과적합의 문제가 발생한다. (degree 값이 커질수록 학습 데이터에만 너무 맞춘 학습이 이뤄져 정작 테스트 데이터 환경에서는 오히려 예측 정확도 ↓)
- 편향-분산 트레이드 오프(Bias-Variance Trade off)
 - 고편향: 매우 단순화된 모델
 - 고분산: 매우 복잡한 모델, 지나치게 높은 변동성
- 최적 모델을 위한 비용함수 구성요소: 학습 데이터 잔차 오류 최소화 + 회귀 계수 크기 제어

5-2. Bias-Variance Trade off

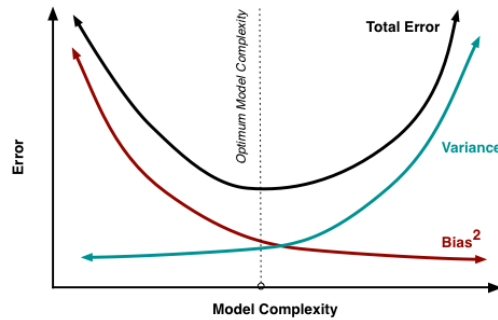
편향-분산 트레이드오프는 머신러닝이 극복해야하는 가장 중요한 이슈 중의 하나



위 예제의 Degree 1과 같은 모델은 매우 단순한 모델로 지나치게 한 방향으로 치우친 경향이 있는 모델이다. 이런 모델을 고편향(High Bias)성을 가졌다고 한다. 반대로 Degree 15와 같은 모델은 학습 데이터 하나하나의 특성을 매우 잘 반영하면서 지나치게 복잡한 모델이 되었고 높은 변동성을 가지게 되었다. 이런 모델은 고분산(High Variance)성을 가졌다고 말한다.



- Low Bias/Low Variance는 예측 결과가 실제 결과에 비해 매우 잘 근접하면서도 예측 변동이 크지 않고 특정 부위에 집중돼 있는 아주 뛰어난 성능을 보여준다.
- Low Bias/High Variance는 예측 결과가 실제 결과에 비교적 근접하지만, 예측 결과가 실제 결과를 중심으로 꽤 넓게 분포되고 있다.
- High Bias/Low Variance는 정확한 결과에서 벗어나면서도 예측이 특정 부분에 집중되어 있다.
- High Bias/High Variance는 정확한 예측 결과를 벗어나면서도 넓은 부분에 분포되어 있다.



일반적으로 bias와 variance는 한 쪽이 높으면 다른 한 쪽이 낮아지는 경향이 있다. bias가 너무 높으면 전체 error가 높고, bias를 점점 낮추면 동시에 variance는 높아지면서 전체 error는 낮아진다. 그리고 전체 error가 가장 낮아지는 'Goldilocks' 지점을 통과하면서 분산은 지속적으로 높이면 전체 error는 오히려 증가하면서 예측 성능이 다시 낮아진다. high bias/low variance는 과소적합되기 쉬우며 low bias/high variance는 과적합되기 쉽다. bias와 variance가 서로 trade off를 이루면서 가장 낮은 Cost를 가지는 모델을 구축하는 것이 가장 효율적인 머신러닝 예측 모델이다.

6 규제 선형 모델 - 릿지, 라쏘, 엘라스틱넷

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

alpha를 0에서부터 지속적으로 값을 증가시키면 회귀 계수 값의 크기를 감소시킬 수 있다. 이처럼 비용 함수에 alpha 값으로 패널티를 부여해 회귀 계수 값의 크기를 감소시켜 과적합을 개선하는 방식을 규제(Regularization)라고 부른다.

규제는 크게 L2 방식과 L1 방식으로 구분된다. L2 규제는 위와 같이 $\alpha * ||W||_2^2$ 를 부여하는 하는 방식이며, L2 규제를 적용한 회귀를 릿지(Ridge) 회귀라고 한다. L1 규제는 $\alpha * ||W||_1$ 를 부여하는 방식이고, L1 규제를 적용한 회귀는 라쏘(Lasso) 회귀라고 부른다. L1 규제를 적용하면 영향력이 크지 않은 회귀 계수를 0으로 만든다.

릿지 회귀(L2)

w^2 에 대해 패널티를 부여하는 방식. 주요 생성 파라미터는 alpha로, alpha가 커질 수록 회귀 계수 값을 작게 만든다.

$$RSS(w) + \alpha * ||w||_2^2 \text{ 식을 최소화하는 } w \text{를 찾는 것}$$

```
from sklearn.linear_model import Ridge
```

라쏘 회귀(L1)

$|w|$ 에 패널티를 부여하는 방식. 불필요한 회귀 계수를 급격하게 감소시켜 0으로 만든다.

$$RSS(w) + \alpha * ||w||_1 \text{ 식을 최소화하는 } w \text{를 찾는 것}$$

```
from sklearn.linear_model import Lasso
```

엘라스틱넷 회귀

- L2규제와 L1규제를 결합한 회귀
 - L1 라쏘 회귀가 중요 피처만 선택하고 다른 피처들은 회귀 계수를 0으로 만드는 성향이 강함
 - alpha 값에 따라 회귀 계수 값이 급격히 변동할 수 있는데, 이를 완화하기 위해 L2 릿지를 L1 라쏘 회귀에 추가한 것
 - 반대로 엘라스틱넷 회귀의 단점은 L1 + L2로 수행시간이 상대적으로 오래 걸린다는 것

$RSS(w) + \alpha_1 * ||w||_1 + \alpha_2 * ||w||_2^2$ 식을 최소화하는 w 를 찾는 것

- 엘라스틱 규제 : $a*L1+b*L2$
- 릿지와 라쏘의 alpha값과는 다름
- $a = L1$ 규제의 alpha값, $b = L2$ 규제의 alpha값
- 엘라스틱넷 회귀의 alpha값은 $= a+b$
- 주요 생성 파라미터
 - `alpha` = $a+b$
 - `l1_ratio` = $(a / (a+b))$
 - `l1_ratio = 0` 이면, $a=0$ 이므로 L2규제와 동일
 - `l1_ratio = 1` 이면, $b=0$ 이므로 L1 규제와 동일

```
from sklearn.linear_model import ElasticNet
```

선형 회귀 모델을 위한 데이터 변환

- 데이터 분포도의 정규화와 인코딩의 중요성
- 데이터 변환 : 선형 회귀 모델은 피처값과 타겟값의 분포가 정규분포로 된 형태를 선호
 - 정규분포 형태가 아니라, 특정값 분포로 치우친 왜곡(Skew)된 형태의 분포도일 경우, 성능에 부정적 영향 줄 가능성 높음
- 1. StandardScaler : 평균이 0, 분산이 1인 정규분포를 가진 데이터 세트로 변환
MinMaxScaler : 최소값이 0, 최대값이 1인 값으로 정규화 수행
- 2. 스케일링/정규화를 수행한 데이터 세트에 다시 다항 특성을 적용하여 변환
 - 1번 방법을 통해 성능 향상이 없는 경우, 2번을 적용하는 경우가 많음
- 3. ★로그 변환★ : log함수를 적용하여 정규분포에 가까운 형태로 변환
 - 이 방법을 주로 사용
 - 1번은 성능 향상을 크게 기대하기 어려우며

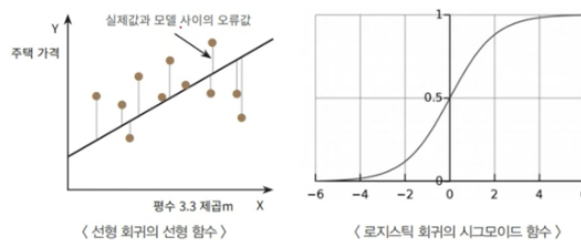
- 2번은 피쳐 갯수가 매우 많을 경우에는 다항 변환으로 생기는 피쳐 갯수가 기하급수로 늘어나서 과적합 이슈 발생 높음
- 타깃값의 경우는 일반적으로 로그 변환을 적용 → 타깃값을 다른류의 정규값으로 변환하면 원복이 어렵고, 왜곡된 타깃 분포도를 로그 변환하면 성능 향상된 경우가 많은 사례에서 검증됐음
- `np.log1p()` = $1+\log()$ 사용
 - $\log()$ 는 언더플로우 발생 가능성이 있기 때문

Polynomial은 값이 잘 나왔으나 과적합일 수 있음. 보통 로그변환만 하는 추세

7 로지스틱 회귀



선형 회귀 방식을 기반으로 하되 시그모이드 함수를 이용해 분류를 수행하는 회귀



- 시그모이드 함수 : $y = \frac{1}{1+e^{-x}}$
 - $x \rightarrow \infty : y = 1$
 - $x \rightarrow -\infty : y = 0$
 - $x \rightarrow 0 : y = 0.5$

- 로지스틱 회귀 하이퍼 파라미터
 - `penalty` : 규제 유형 설정. 'l2', 'l1' (기본은 l2)
 - `c` : 규제 강도 조절하는 alpha의 역수 ($1/\alpha$) ∴ 작을수록 규제강도 높아짐

8 회귀 트리

: 회귀를 위한 트리를 생성하고 이를 기반으로 회귀 예측 진행

: 리프 노드에 속한 데이터 값의 평균값으로 회귀 예측값을 계산

: 결정트리, 랜덤포레스트, GBM, LightGBM, XGBoost 등 모든 트리 기반 알고리즘은 회귀 계산 가능

(뒤에 Regressor. 예: DecisionTreeRegressor)

단, 선형 회귀와 다른 처리 방식이므로 회귀 계수를 제공하는 coef 속성은 없다. 대신 피쳐별 중요도를 알려주는 `feature_importances` 제공