선형 회귀

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

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

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()
```

```
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")
plt.show()
```

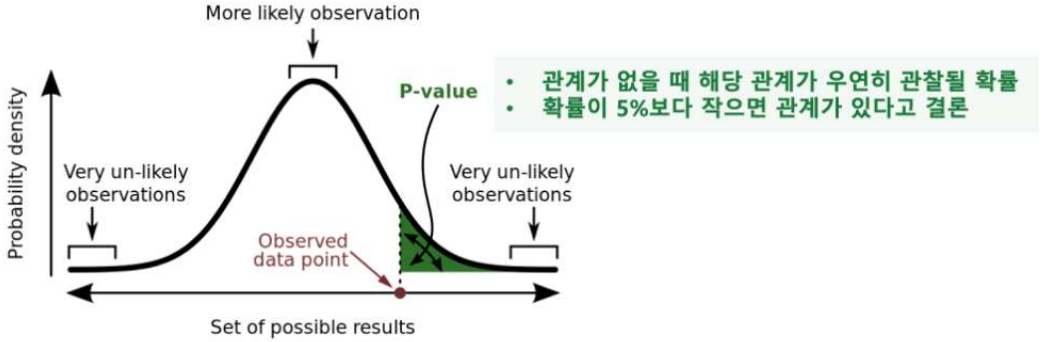
선형 회귀

➢ 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

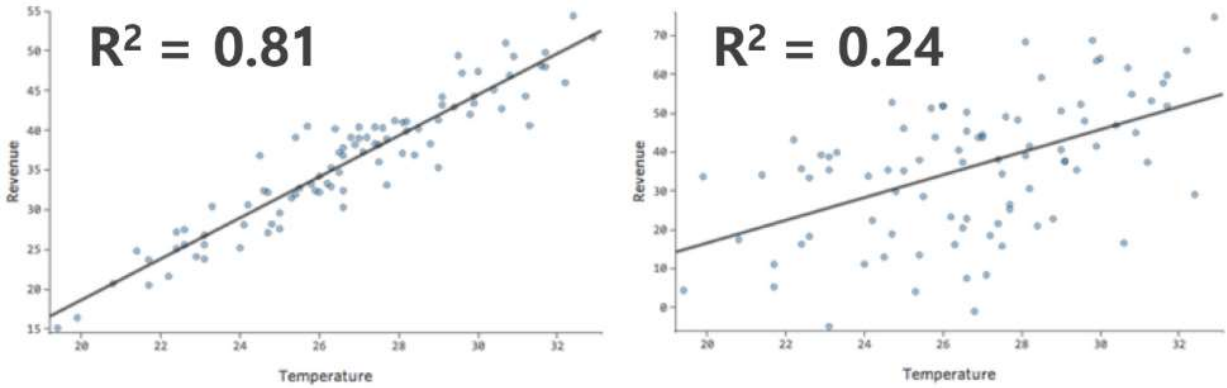
$$Y = b0 + b1X + error$$

- 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

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

➢ 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

➢ 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 않게되거나 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 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
```