Regressor → **연속적인 수치 예측** (무게, 가격, 온도 등)

Classifier → **범주 분류** (종류, 레이블 등)

LinearRegression -> 선형 회귀

LogisticRegression > 로지스틱 회귀

표준점수 : 훈련 세트의 스케일을 바꾸는 대표적인 방법 = 표준화

* (array - mean) / std = 표준화: 모든 특성의 평균이 0, 표준편차가 1이되게 함
* mean = 평균 / std = 표준편차
* 표준편차 = 데이터가 표준(평균 값)에 얼마나 떨어져 있는지 나타내는 지표

브로드캐스팅 : 크기가 다른 넘파이 배열에서 자동으로 사칙 연산을 모든 행이나 열로 확장하여 수행하는 기능

| **회귀** | **분류** |
| --- | --- |
| 숫자를 예측 | 범주(종류)를 예측 |
| 연속적인 값 | 정해진 클래스 |
| 예: 무게, 가격, 온도 | 예: 암/정상, 스팸/일반 |

숫자예측 or 정답

회귀 : 임의의 수치를 예측하는 문제

결정계수(R\*\*2) : 대표적인 회귀 문제의 성능 측정 도구 1에 가까울수록 좋은 모델

과대적합

* 훈련 세트에서 접수가 굉장히 좋았는데 테스트 세트에서는 점수가 나쁜 모델

과소적합

* 반대로 훈련 세트보다 테스트 세트의 저뭇가 두 점수가 모두 너무 낮은 경우

선형 회귀 : 특성과 타깃 사이의 가관계를 가장 잘 나타내는 선형 방정식을 찾는다

모델 파라미터 : 다항식을 사용하여 특성과 타깃 사이의 관계를 나타냄

다항 회귀 : 다항식을 사용하여 특성과 타깃 사이의 관계

**matplotlib**

import matplotlib.pyplot as plt

scatter() : 산점도를 그리는 맷플롯립 함수

* scatter(x, y[, marker]) x, y는 배열도 가능

2차원 배열일 경우 x = array[:,0], y=array[:1]

* marker = ‘^’ : 삼각형 , ‘D’ 사각형

xlabel(), ylabel() : 라벨이름 정하기

* xlabel(“이름”) / ylabel(“이름”)

show() : 보여줌

plt.plot([x₁, x₂], [ax₁ + b, ax₂+b])

* (x₁, lr.coef\_\*x₁ + lr.intercept\_) 부터(x₂, lr.coef\_\*x₂ + lr.intercept\_)좌표까지 직선을 표시

선형 회귀 모델

* x = np.arange(15, 50) 15~49까지 넣겠다
* plt.plot(x, lr.coef\_[0]\*x\*\*2 + lr.coef\_[1]\*x + lr.intercept\_)
* plt.plot(x, ax\*\*2 + bx + c)

**scikit-learn** - 머신러닝 패키지

KNeighborsClassifier() : k–최근접 이웃 분류 모델을 만드는 사이킷런 클래스

* kn = KNeighborsClassifier()

fit() : 사이킷런 모델을 훈련할 때 사용

* kn.fit(input, target) : 훈련

predict() : 사이킷런 모델을 훈련하고 예측할 때 사용 / 값을 넣으면 예측

* kn.predict(test\_input) : 훈련된 모델을 근거로 값을 출력

score() : 훈련된 사이킷런 모델의 성능을 예측할 때 사용

* kn.score(test\_input, test\_target) : 훈련된 모델이 얼마나 잘 맞추는지 확인
* 0 ~ 1까지 1에 가까울 수록 잘된것

train\_test\_split() : 훈련 데이터를 훈련 세트와 테스트 세트로 나누는 함수

* from sklearn.model\_selection import train\_test\_split
* train\_input, test\_input, train\_target, test\_target = train\_test\_split(

data, target[, stratify = target, random\_state = 42])

* stratify = 데이터 비율에 맞게 나눔

전체 2:1이면 2개의 target에게 각각 1:2, 1:2가 되도록 만듬

* random\_state : seed값을 고정

kneighbors() : k-최근접 이웃 객체의 메서드

* distances, indexes = kneighbors([[x, y]])

x, y를 기준으로 가까운 5개? 를 가지고 온다

array.reshape() : 행,열을 다르게 만들어 줌

* array.reshape(row, column) : 행과 열을 만든다 (전체값에 맞아야함)

-1은 전체 / 열은 1 = 2차원

예) array.reshape(-1, 1) 1열로 이루어진 2차원 배열

KNeighborsRegressor() : 회귀 모델

* k-최근접 이웃 회귀 모델을 만드는 사이킷런 클래스입니다. 기본값 5
* n\_neighbors = 3 < 3으로 바뀜
* KNeighborsRegressor(n\_neighbors = 3) < 이런식 사용 가능

mean\_absolute\_error():

* from sklearn.metrics import mean\_absolute\_error
* mean\_absolute\_error(실제값, 예상값)
* 테스트 세트에 대한 평균 절댓값 오차를 계산하다

lr = LinearRegression(): 선형회귀를 말하는 것이다

* from sklearn.linear\_model import LinearRegression

lr.coef\_ : 기울기 / lr.intercept\_ : 절편

PolynomialFeatures(): 주어진 특성을 조합하여 새로운 특성을 만듦

* poly = PolynomialFeatures(degree=, include\_bias=)
* - degree: 최고 차수 (default=2 → 제곱까지)
* - include\_bias: 상수항 1을 포함할지 여부 (default=True → 포함함)
* poly.fit([[a, b]])
* - fit을 먼저 해야 transform 가능
* poly.transform([[a, b]])
* - fit 시 사용한 특성과 같은 열 개수여야 transform 가능
* 변환 결과 예시 >> [1, a, b, a^2, a\*b, b^2]
* (shape은 (x, 6) ← x는 샘플 개수)
* 문제 너무 과도하게 학습 - 규제필요

StandardScaler(): 특성 표준화 도구

* x' = (x - 평균) / 표준편차
* → 평균 = 0, 표준편차 = 1로 만들어줌
* → 특성 간 스케일 통일
* ss = StandardScaler()
* ss.fit(data) # 데이터의 평균, 표준편차 학습
* scaled = ss.transform(data) # 표준화 적용
* # 또는 한 줄로:
* scaled = ss.fit\_transform(data)
* ⚠️ 주의: fit은 훈련 데이터에만! 테스트 데이터엔 transform만 사용

Ridge() : 릿지 회귀

* from sklearn.linear\_model import Ridge
* ridge = Ridge()
* ridge.fit(scaled, target)
* alpha는 그 **벌점의 세기**예요
* 값이 작을수록 규제가 약해짐 모델이 자유롭게 학습함 ->과적합 위험
* 클수록 규제가 강해짐 모델이 너무 단순해짐 0> 과소적합

alpha\_list = [0.001, 0.01, .0.1, 1, 10, 100]

alpha : 규제(별점)의 세기를 조절하는 하이퍼파라미터

for i in alpha\_list:

ridge = Ridge(alpha = i)

ridge.fit(train\_scaled, train\_target)

train\_score.append(ridge.score(train\_scaled, train\_target))

test\_score.append(ridge.score(test\_scaled, test\_target))

>규제를 다르게 해서 모델 점수를 각각의 리스트에 넣기

plt.plot(alpha\_list, train\_score)

plt.plot(alpha\_list, test\_score)

plt.xscale(‘log’)

plt.xlabel(‘alpha’)

plt.ylabel(‘R^2’)

2개의 리스트들을 확인하고 test값이 크고 train값이 test 모델 점수의 차이가 적은걸 갖기 위함

✅ Ridge 회귀 모델 성능 비교 (alpha 값 변화에 따른 R^2 확인)

🔹 alpha\_list = [0.001, 0.01, 0.1, 1, 10, 100]

→ alpha: 규제(벌점)의 세기를 조절하는 하이퍼파라미터

🔹 목적:

- 규제를 다르게 하여 모델을 학습시킨 후

- 훈련(train) 및 테스트(test) 데이터의 R^2 점수를 비교

- 일반화 성능이 좋은 alpha 값을 찾기 위함

🔹 좋은 모델 조건:

- test 점수가 높고

- train과 test 점수 차이가 작을 것

Lasso():

* lasso.fit(train\_scaled, train\_target) < 라쏘 모델 학습

alpha\_list = [0.001, 0.01, 0.1, 1, 10, 100]

for alpha in alpha\_list:

lasso =Lasso(alpha = alpha, max\_iter = 10000)

max\_iter=10000: 반복 횟수 설정 (라쏘는 수렴이 느릴 수 있어서 여유 있게 설정)

* max\_iter 의 숫자 안으로 최적의 가중치를 찾는것

lasso.fit(train\_scaled, train\_target) < 안에 10000반복(무조건은 아님)

train\_score.append(lasso.score(train\_scaled, tdrain\_target))

test\_score.append(lasso.score(test\_scaled, test\_target))

| **비교** | **릿지(Ridge)** | **라쏘(Lasso)** |
| --- | --- | --- |
| 벌점 방식 | w를 **작게만** 만듦 | w를 **0으로도 만듦** |
| 의미 | “다 살짝 줄여~” | “필요 없는 건 아예 없애!” |
| 결과 | 모든 특성을 조금씩 사용함 | **쓸모없는 특성은 제거함** |
| 특성 선택 | ❌ 못함 | ✅ **가능함** |
| 쓰는 상황 | 특성 다 중요할 때 | **특성이 많고 일부만 중요할 때** |

**numpy**

import numpy as np

seed() : 넘파이에서 난수를 생성하기 위한 정수 초깃값을 지정

arange() : 일정한 간격의 정수 또는 실수 배열을 만듬

* np.arange([시작,] 끝[, 간격])

shuffle() : 주어진 배열을 랜덤하게 섞음

* np.random.shuffle(array)

column\_stack() : 행과 열을 바꾸어 배열을 만들어줌

* column\_stack(array1, array2)

**pandas**

import pandas as pd

pd.read\_csv() : csv파일을 로컬 컴퓨터나 인터넷에서 읽어 판다스

데이터프레임으로 변환하느 함수