5장 회귀

# Ch	6
날짜	@2025년 10월 12일
카테고리	개념 정리

5.1 회귀 소개

: **연속형 데이터를 예측**하는 지도학습 기법

; 목표는 주어진 데이터로부터 **최적의 회귀 계수(Weight)** 를 찾는 것

- **유래**
  - **갈톤(Galton)** 의 부모-자식 키 연구에서 유래 →
    자식의 키가 부모보다 크거나 작더라도 **평균 키로 회귀하려는 경향** 발견
  - 회귀 분석은 **데이터 값이 평균으로 돌아가려는 경향을 이용하는 통계학 기법**임
- **정의**
  - **회귀(Regression)**: 여러 개의 **독립변수(X)** 와 한 개의 **종속변수(y)** 간의 상관관계를 모델링하는 기법
  - 예시: 아파트 가격(y) = 방 개수(x_1), 방 크기(x_2), 학군(x_3) 등의 독립변수로 설명

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n$$

  → 머신러닝에서는 X는 **피처(feature)**, y는 **결정값(target)** 에 해당

  → 학습의 목표: **최적의 회귀 계수(Weights)**를 찾아내는 것

- **분류**

구분 기준	유형	설명
독립변수 개수	단일 회귀	독립변수 1개
	다중 회귀	독립변수 여러 개
회귀 계수 형태	선형 회귀	회귀 계수가 선형 결합
	비선형 회귀	비선형 결합

- **분류(Classification) vs 회귀(Regression)**

구분	분류(Classification)	회귀(Regression)
예측값 형태	범주형 (이산값)	수치형 (연속값)
예시	이메일 스팸 여부, 질병 유무	주택 가격, 온도 예측

- **선형 회귀의 개념**
  - 실제값과 예측값의 **오차 제곱합(RSS)** 을 최소화하는 직선형 모델
  - 규제(Regularization)를 적용하지 않으면 일반적인 선형 회귀,
    규제를 적용하면 Ridge/Lasso/ElasticNet으로 구분됨
- **회귀모델 종류**

모델	규제 방식	특징
일반 선형 회귀	없음	RSS 최소화, 기본형
릿지(Ridge)	L2 규제	큰 회귀계수의 크기를 줄임
라쏘(Lasso)	L1 규제	중요하지 않은 피처의 계수를 0으로 만들어 **피처 선택 기능** 수행
엘라스틱넷(ElasticNet)	L1 + L2 혼합	피처 수가 많을 때 유용, 변수 선택 + 크기 조정 동시 수행
로지스틱 회귀(Logistic Regression)	-	회귀라는 이름이지만 실제로는 **분류용 선형 모델** (텍스트 분류 등에서 우수한 성능)

5.2 단순 선형 회귀를 통한 회귀 이해

> key
>
> - 단순 선형 회귀는 **하나의 변수로 연속형 값을 예측**하는 가장 기본적인 회귀 형태
> - 학습의 목표는 **오류 제곱합(RSS)을 최소화**하는 **최적의 회귀 계수(w_0, w_1)** 를 찾는 것
> - RSS는 회귀 모델의 성능을 측정하는 **비용 함수(Cost/Loss Function)** 역할을 함

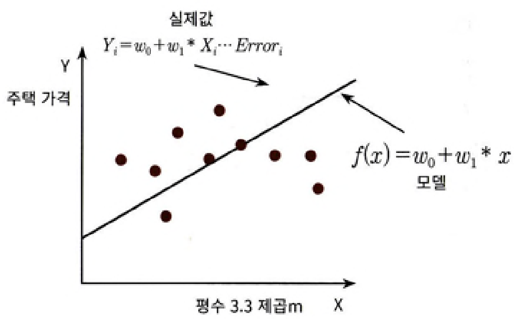
1. 단순 선형 회귀의 개념

- **단순 선형 회귀(Simple Linear Regression)**
  → 독립변수 1개(X)와 종속변수 1개(y) 간의 선형(직선) 관계를 모델링
- 예시:

  주택 가격(y)이 주택 크기(X)에 의해 결정된다고 가정할 때, 크기가 커질수록 가격이 상승하는 경향을 **직선 형태로 표현** 가능

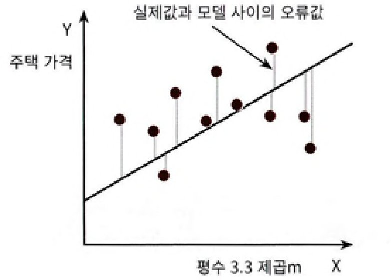
  y=w0+w1X

  - w0w_0w0: 절편(intercept)
  - w1w_1w1: 기울기(slope)
  - w0,w1w_0, w_1w0,w1을 **회귀계수(regression coefficients)**



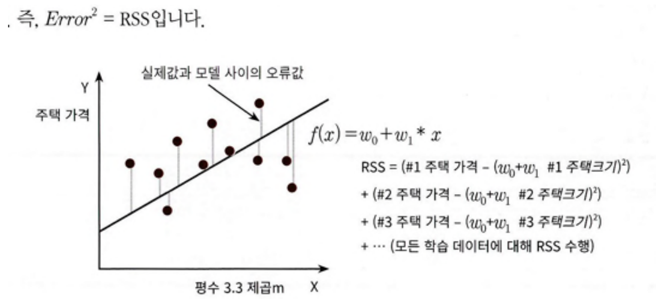
2. 회귀식과 잔차(Residual)

- 실제값은 모델 예측값에서 오차(잔차, residual)가 더해진 형태로 표현됨
  yi=(w0+w1xi)+(오류값)



- **잔차(Residual)**: 실제값과 예측값의 차이
- **목표:** 모든 데이터의 잔차 제곱합을 최소화하는 **최적의 회귀 계수(w_0, w_1)** 찾기

3. 오류(오차) 측정 방식

> 단순 합산 시 +, − 오차가 상쇄될 수 있어,
>
> 일반적으로 **RSS(제곱합)** 방식이 사용됨
>
> 
>

방식	계산식	설명
MAE (Mean Absolute Error)	평균(	실제값 - 예측값
RSS (Residual Sum of Squares)	∑(실제값 - 예측값)²	제곱합 기준 오차 (미분 계산에 유리)

4. 비용 함수(Cost Function)

- RSS는 회귀 계수 w0,w1w_0, w_1w0,w1 에 의해 결정되는 **비용 함수(cost function)**
- 머신러닝 회귀의 핵심은 **RSS를 최소화하는 w_0, w_1을 학습으로 찾는 것**

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

(i는 1부터 학습 데이터의 총 건수 N까지)

- 학습 데이터의 (X, y)는 상수로 간주
- w0,w1w_0, w_1w0,w1이 RSS를 최소화하도록 지속적으로 업데이트됨
- RSS는 **비용 함수(Cost Function)** 또는 **손실 함수(Loss Function)** 라고도 함

5.3 비용 최소화하기 - 경사 하강법

: **비용 함수가 최소가 되는 방향으로 파라미터를 점진적으로 조정**하는 알고리즘

1. 경사 하강법의 개념

> 머신러닝에서 “데이터로부터 스스로 학습한다”는 개념을 가능하게 한 핵심 기법

- **경사 하강법(Gradient Descent)**

  : 비용 함수(RSS)를 최소화하는 **최적의 회귀 계수(w)** 를 찾기 위한 **반복적 최적화 알고리즘**
- 고차원 방정식으로 해석적 계산이 어려운 경우, **데이터를 기반으로 점진적으로 학습하며 최소값을 탐색**

2. 비유를 통한 이해...

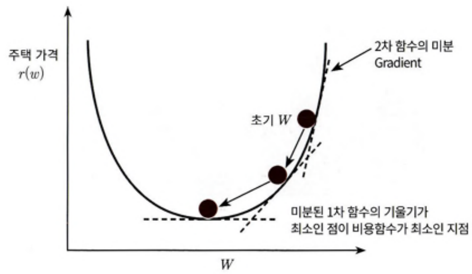
- 깜깜한 산 정상에서 **가장 낮은 지점(최소 비용)** 을 향해 한 걸음씩 **내리막 방향(오류가 줄어드는 방향)** 으로 내려가는 과정
- 매번 현재 위치에서 기울기(gradient)를 계산하고, 그 반대 방향으로 **w 값을 업데이트(update)** 하며 비용을 점점 줄임

3. 작동 원리

1. 초기 임의의 www 설정
2. **비용 함수(Cost Function)** 의 기울기(미분값)를 계산
3. 기울기의 반대 방향으로 www 를 이동 (오류 감소 방향)
4. 더 이상 비용이 줄어들지 않으면, 그 시점의 w 를 **최적 파라미터**로 판단

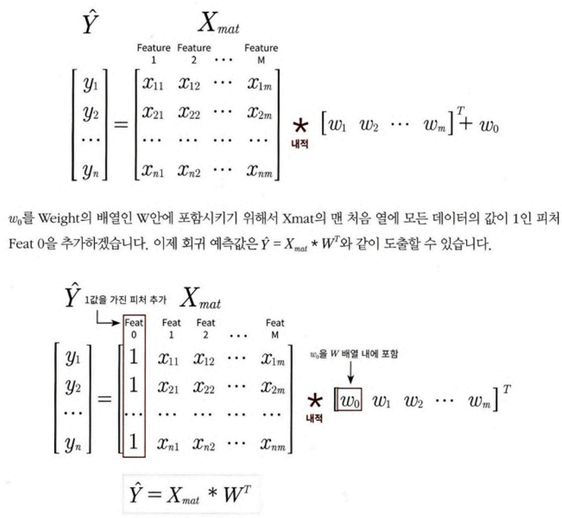
- **기울기 = 변화율 = 미분값**
- 비용 함수가 포물선 형태라면, 미분값(기울기)이 0이 되는 지점이 **비용의 최소점(minimum)**
- 경사 하강법은 이 원리를 이용해 반복적으로 최소점을 찾음

<포물선 형태 2차 함수에서의 경사 하강법>



<피처가 여러 개인 경우>

- 피처가 한 개의 경우의 예측값으로 회귀 계수 도출
  ⇒ 피처가 M개 있다면 그에 따른 회귀 계수도 M+1개로 도출



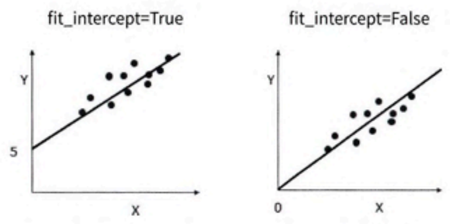
5.4 사이킷런 LinearRegression을 이용한 보스턴 주택 가격 예측

LinearRegression 클래스 - Ordinary Least Squares

> LinearRegression 클래스는 예측값과 실제값의 RSS(Residual Sum of Squares) 를 최소화하는 **OLS(Ordinary Least Squares, 최소제곱법)** 기반의 회귀 모델입니다.

- `fit(X, y)` 실행 시 회귀 계수(W)를 `coef_` 속성에 저장
- **입력 피처 간의 독립성**이 중요하며, 피처 간 상관관계가 높으면 **다중공선성 문제(Multicollinearity)** 발생
- 다중공선성 해결 방법
  - 상관관계가 높은 피처 제거
  - 규제(Regularization) 적용 (Ridge, Lasso 등)
  - **PCA(주성분 분석)** 을 통한 차원 축소

```
class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)
```

입력 파라미터	**fit_intercept**: 불린 값으로, 디폴트는 True입니다. Intercept(절편) 값을 계산할 것인지 말지를 지정합니다. 만일 False로 지정하면 intercept가 사용되지 않고 0으로 지정됩니다.  **normalize**: 불린 값으로 디폴트는 False입니다. fit_intercept가 False인 경우에는 이 파라미터가 무시됩니다. 만일 True이면 회귀를 수행하기 전에 입력 데이터 세트를 정규화합니다.
속성	**coef_**: fit() 메서드를 수행했을 때 회귀 계수가 배열 형태로 저장하는 속성. Shape는 (Target 값 개수, 피처 개수). **intercept_**: intercept 값

회귀 평가 지표(Regression Metrics)

> 회귀의 평가는 실제값과 예측값의 차이를 기반으로 측정합니다.
> 단순 합으로 계산하면 +/– 오차가 상쇄되므로, 절댓값 또는 제곱을 활용합니다.

지표	정의	수식	설명
MAE (Mean Absolute Error)	실제값과 예측값의 차이의 절댓값 평균	MAE = (1/n) Σ	$y_i - \hat{y}_i$
MSE (Mean Squared Error)	실제값과 예측값의 차이의 제곱 평균	MSE = (1/n) Σ $(y_i - \hat{y}_i)^2$	큰 오차에 더 큰 페널티
RMSE (Root Mean Squared Error)	MSE의 제곱근	$RMSE$ = √MSE	실제 단위로 해석 가능
R^2 (결정계수)	예측값의 분산 / 실제값의 분산	R^2 = Var(ŷ) / Var(y)	1에 가까울수록 예측 성능 높음

추가 지표

- **MSLE** (Mean Squared Log Error)
- **RMSLE** (Root Mean Squared Log Error) — 로그를 적용한 RMSE

⚙️ 사이킷런 평가 지표 API & Scoring 파라미터

평가 방법	사이킷런 API	Scoring 함수 값
MAE	`metrics.mean_absolute_error`	`'neg_mean_absolute_error'`
MSE	`metrics.mean_squared_error`	`'neg_mean_squared_error'`
RMSE	`metrics.mean_squared_error(squared=False)`	`'neg_root_mean_squared_error'`
MSLE	`metrics.mean_squared_log_error`	`'neg_mean_squared_log_error'`
R^2	`metrics.r2_score`	`'r2'`

⚠️ Scoring 함수 사용 시 주의점

- `cross_val_score` , `GridSearchCV` 의 `scoring` **파라미터는 "값이 클수록 좋은 모델"** 로 인식
- 하지만 MAE, MSE, RMSE 등은 값이 **작을수록 좋은 지표**이므로 사이킷런은 자동으로 **음수(Negative)** 를 붙여 평가

```
-1 * metrics.mean_absolute_error(y_true, y_pred)
```

LinearRegressionS 이용해 보스턴 주택 가격 회귀 구현

1. 데이터 준비

- **Boston Housing Dataset** 사용
- 모든 feature는 `float` 형, **결측치 없음** (`bostonDF.info()` 로 확인 가능)

2. 피처별 PRICE 영향 시각화

- 주요 피처: `RM` , `ZN` , `INDUS` , `NOX` , `AGE` , `PTRATIO` , `LSTAT`
- `seaborn.regplot()` 을 이용해 산점도 + 회귀직선 시각화

> **<시각화 결과 해석>**
>
> - **RM (방 개수) ↗ → PRICE 증가 (양의 선형성)**
> - **LSTAT (하위 계층 비율) ↘ → PRICE 감소 (음의 선형성)**
>
>   ➡️ 두 변수가 **PRICE에 가장 큰 영향**

3. 선형 회귀 모델 학습

```
# 학습/테스트 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=156
)

# 모델 학습
lr = LinearRegression()
lr.fit(X_train, y_train)

# 예측
y_pred = lr.predict(X_test)
```

4. 성능 평가

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R²: {r2:.4f}")
```

5.5 다항 회귀와 과적합/과소적합

다항 회귀 이해

- 일반적인 **선형 회귀**는 **직선 관계**로 모델링함.

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

- 그러나 현실의 데이터는 **직선으로 설명되지 않는 경우**가 많음.
  → 이를 **2차, 3차 다항식 형태**로 확장한 것이 **다항 회귀**

$$y = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots$$

PolynomialFeatures로 다항식 피처 생성

> `PolynomialFeatures(degree=2)`
> → 입력 [x_1, x_2] → [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]로 확장

```
from sklearn.preprocessing import PolynomialFeatures
import numpy as np

# 다항식으로 변환한 단항식 생성, [[0, 1], [2, 3]]의 2X2 행렬 생성
X=np.arange(4).reshape(2,2)
print('일차 단항식 계수 피처:\n',X)

# degree = 2인 2차 다항식으로 변환하기 위해 PolynomialFeatures를 이용해 변환
poly=PolynomialFeatures(degree=2)
poly.fit(X)
poly_ftr=poly.transform(X)
print('변환된 2차 다항식 계수 피처:\n',poly_ftr)
```

```
일차 단항식 계수 피처:
[[0 1]
 [2 3]]
변환된 2차 다항식 계수 피처:
[[1. 0. 1. 0. 0. 1.]
 [1. 2. 3. 4. 6. 9.]]
```

```
# 3차 다항식 변환
poly_ftr=PolynomialFeatures(degree=3).fit_transform(X)
print('3차 다항식 계수 feature:\n',poly_ftr)

# Linear Regression에 3차 다항식 계수 feature와 3차 다항식 결정값으로 학습 후 회귀 계수 확인
model=LinearRegression()
model.fit(poly_ftr,y)
print('Polynomial 회귀 계수\n',np.round(model.coef_,2))
print('Polynomial 회귀 Shape:',model.coef_.shape)
```

```
3차 다항식 계수 feature:
 [[ 1.  0.  1.  0.  0.  1.  0.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.  8. 12. 18. 27.]]
Polynomial 회귀 계수
 [0.   0.18 0.18 0.36 0.54 0.72 0.72 1.08 1.62 2.34]
Polynomial 회귀 Shape: (10,)
```

다항 회귀를 이용한 과소적합 / 과적합 이해

단계	설명
`true_fun(X)`	실제 데이터의 기본 함수로 코사인 곡선 생성
`PolynomialFeatures`	다항식 특성을 자동 생성 (degree만 바꾸면 됨)
`Pipeline`	다항 변환 + 선형 회귀 과정을 묶어서 간단히 실행
`cross_val_score`	교차검증으로 MSE 계산 (neg_mean_squared_error 사용)
`X_test`	예측용 데이터 (0~1 구간에서 100개 점)
`plt.subplot()`	세 개의 그래프를 1행 3열 형태로 시각화
`degree`	1 → 과소적합 / 4 → 적절 / 15 → 과적합 패턴 확인

실행 결과 해석

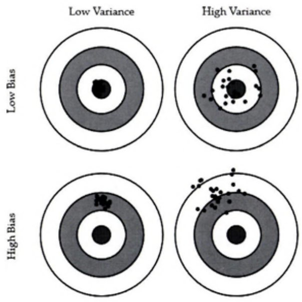
Degree	MSE (평균제곱오차)	모델 특징
1	약 0.4	너무 단순해서 패턴을 못 잡음 → **과소적합**
4	약 0.04	코사인 패턴을 잘 잡음 → **적절한 모델**
15	매우 큼 (예: 1.8e+08)	잡음까지 학습 → **과적합**

- Degree 15의 회귀 계수 값이 Degree 1, 4와 비교할 수 없을 정도로 매우 큰 값

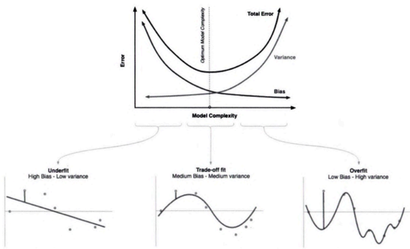
  ⇒ Degree 15라는 복잡한 다항식 만족하기 위해 계산된 회귀 계수는 결국 현실과 동떨어진 예측 결과 초래함

- 좋은 모델은 Degree 1과 같이 학습 데이터 패턴 지나치게 단순화한 과소적합 모델도 아니고, Degree 15와 같이 모든 학습 데이터 패턴 하나하나 감안한 과적합 모델도 아닌
  → **학습 데이터 패턴 잘 반영하면서도 복잡하지 않은 균형 잡힌 모델 의미**

편향 - 분산 트레이드 오프 (Bias-Variance Trade Off)



〈 편향과 분산의 고/저에 따른 표현.
http://scott.fortmann-roe.com/docs/BiasVariance.html에서 발췌 〉



〈 편향과 분산에 따른 전체 오류 값(Total Error) 곡선. http://scott.fortmann-roe.com/docs/BiasVariance.html에서 발췌. 〉

용어	설명	예시
편향(Bias)	모델이 실제 데이터 패턴을 얼마나 잘 학습하지 못했는지, 즉 예측의 평균이 실제 값과 얼마나 차이가 나는지	단순한 직선으로 복잡한 데이터를 모델링할 때 → 과소적합
분산(Variance)	모델이 학습 데이터의 작은 변동에도 얼마나 민감하게 반응하는지	너무 복잡한 모델이 학습 데이터의 잡음까지 반영할 때 → 과적합

$$\text{전체 오류(Total Error)} = \text{편향}^2 + \text{분산} + \text{잡음(Noise)}$$

- **편향이 높으면 분산은 낮음**
  → 모델이 단순하고 일정하게 예측하지만 실제 값과 큰 차이
  ⇒ 과소적합
- **분산이 높으면 편향은 낮음**
  → 모델이 학습 데이터에는 잘 맞지만 테스트 데이터에는 불안정
  ⇒ 과적합
- **목표**: 편향과 분산의 균형을 맞춰 전체 오류(Total Error)를 최소화
- 편향이 너무 높으면 전체 오류가 크고, 분산을 높이면 오류가 낮아지다가 일정 수준 이후에는 다시 증가 → **골디락스 (Goldilocks) 지점**에서 오류 최소

머신러닝 적용 시

- **과소적합** → 모델이 너무 단순 (높은 편향, 낮은 분산)
- **과적합** → 모델이 너무 복잡 (낮은 편향, 높은 분산)
- **좋은 모델** → 적절한 편향과 분산의 균형 → 전체 오류 최소화

> 핵심: 편향과 분산은 서로 트레이드오프 관계에 있으므로, 모델 복잡도를 조절하여 오류를 최소화하는 것이 목표

5.6 규제 선형 모델 - 릿지 라쏘 엘라스틱넷

> key
>
> - 규제는 **회귀 계수를 제어**해 과적합을 방지하는 기법
> - **Ridge(L2)** → 계수 감소, 0으로 만들지 않음
> - **Lasso(L1)** → 일부 계수 0으로 만들며 피처 선택 가능
> - **ElasticNet(L1+L2)** → Lasso의 불안정성 보완

> - 데이터 변환(정규화, 로그 변환) → 모델 성능 향상에 필수
> - 최적 모델 선택 → **alpha 하이퍼파라미터 + 데이터 전처리 조합**에 따라 결정

규제 선형 모델의 개요

- **문제 상황**
  - Degree 1 다항 회귀 → 과소적합(underfitting), 데이터 패턴을 잘 반영하지 못함
  - Degree 15 다항 회귀 → 과적합(overfitting), 회귀 계수(W)가 너무 커짐, 테스트 성능 저하
- **해결 방법**
  - 회귀 계수 W를 적절히 제어하여 과적합 방지
  - 비용 함수(Cost Function) 변경:

$$\text{Cost} = \text{RSS}(W) + \alpha \cdot ||W||_p$$

  - 비용 함수 목표:

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

- **alpha**: 학습 적합도와 회귀 계수 크기 제어 균형을 조절하는 하이퍼파라미터
  - alpha = 0 → 기존 RSS 최소화, 규제 없음
  - alpha → 매우 큼 → W를 작게 만들어 과적합 방지
- **규제 종류**
  - **L2 규제(Ridge)**:

$$\alpha||W||_2^2, \text{ 회귀 계수 크기 감소, 0으로 만들지 않음}$$

  - **L1 규제(Lasso)**:

$$\alpha||W||_1, \text{ 일부 회귀 계수를 0으로 만들어 피처 선택 효과}$$

  - **ElasticNet**: L1 + L2 규제 결합, Lasso의 단점을 보완

릿지 회귀

- **L2 규제 적용**
- **특징**
  - 회귀 계수 값을 작게 만들어 과적합 방지
  - 회귀 계수를 0으로 만들지는 않음
- **실습 예제**
  - alpha = [0, 0.1, 1, 10, 100]
  - 5폴드 교차검증(CV)으로 RMSE 측정
  - alpha 증가 → 회귀 계수 감소

- 예시 결과: alpha=100 → 평균 RMSE 5.330 (규제 없는 LinearRegression 5.829보다 개선)

라쏘 회귀

- **L1 규제 적용**
- **특징**
  - 일부 회귀 계수를 0으로 만들어 불필요한 피처 제거
  - 피처 선택 효과
- **실습 예제**
  - alpha = [0.07, 0.1, 0.5, 1, 3]
  - alpha 증가 → 일부 피처 계수 0으로 변환
  - 예시 결과: alpha=0.07 → 평균 RMSE 5.612 (LinearRegression 5.829보다 개선)

엘라스틱넷 회귀

- **L1 + L2 규제 결합**
- **특징**
  - Lasso의 피처 선택 기능과 Ridge의 안정성 결합
  - 상관관계 높은 피처 문제 완화
  - 단점: 수행시간 증가
- **실습 예제**
  - alpha = [0.07, 0.1, 0.5, 1, 3], l1_ratio=0.7 고정
  - alpha=0.5 → RMSE 5.467, 일부 피처 계수는 0이지만 Lasso보다 적음

선형 회귀 모델을 위한 데이터 변환

- **목적:** 선형 모델은 피처와 타깃이 선형 관계, 정규분포를 선호
- **변환 방법**
  1. **StandardScaler** → 평균 0, 분산 1
  2. **MinMaxScaler** → 0~1 정규화
  3. **PolynomialFeatures** → 다항식 특성 추가 (degree ≤ 2)
  4. **Log Transformation** → skewed 타깃값/피처 정규화
- **실습 결과 요약**
  - 로그 변환(Log) 적용 시 RMSE 개선 가장 뚜렷
  - 다항 변환은 피처 수 증가 → 과적합 위험, 계산 비용 증가
  - alpha 값과 데이터 변환 조합에 따라 최적 RMSE 달성 가능

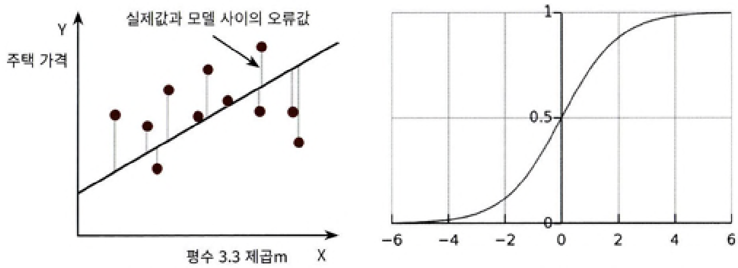
5.7 로지스틱 회귀

> **요약**
>
> 1. 로지스틱 회귀는 **시그모이드 함수를 이용한 확률 기반 이진 분류 모델**
> 2. solver는 `liblinear`(소규모 데이터), `lbfgs`(대규모 데이터)가 주로 사용됨

> 3. 규제(Regularization)로 **L1, L2**를 적용 가능
> 4. 하이퍼파라미터 `C`는 규제 강도 조절용
> 5. GridSearchCV를 이용해 solver, penalty, C를 최적화 가능
> 6. 텍스트 분류 등 희소 데이터에도 **성능이 우수하고 빠름**

1. 개요

- 로지스틱 회귀는 **선형 회귀를 분류 문제에 적용한 알고리즘**
- 이름은 "회귀"지만, 실제로는 **분류(classification)**에 사용됨
- 선형 회귀와의 차이점은, 단순히 선형식을 예측하는 것이 아니라
  시그모이드(Sigmoid) 함수를 통해 확률값을 계산하고, 그 확률을 기반으로 **이진 분류**(0 또는 1)를 수행



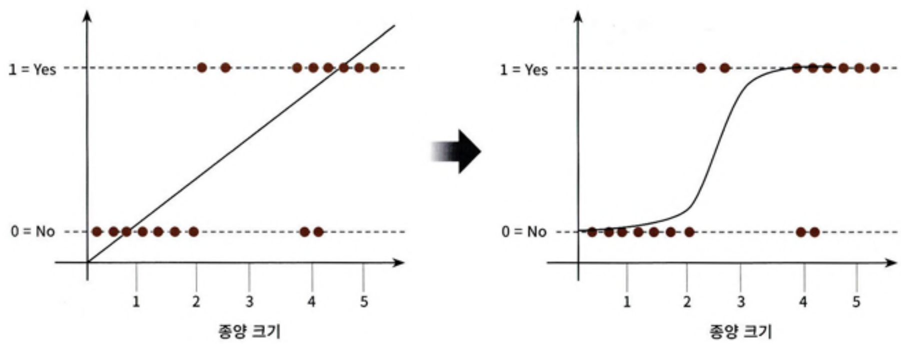
〈선형 회귀의 선형 함수〉 〈로지스틱 회귀의 시그모이드 함수〉

2. 시그모이드 함수

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

- x가 커지면 y → 1
- x가 작아지면 y → 0
- x = 0일 때 y = 0.5

즉, 입력값이 커질수록 "1일 확률", 작을수록 "0일 확률"로 해석
→ 로지스틱 회귀는 "특정 사건이 일어날 확률"을 예측



3. 사이킷런의 LogisticRegression 클래스

solver	설명
lbfgs	기본값. 메모리 효율적이며 병렬 처리 가능
liblinear	작은 데이터셋에서 빠르고 효과적. 단, 병렬처리 불가
newton-cg	정교하지만 느림
sag	확률적 평균 경사하강법. 대용량 데이터에 빠름
saga	sag 기반, L1 규제 가능

실습 - 위스콘신 유방암 데이터 이진분류

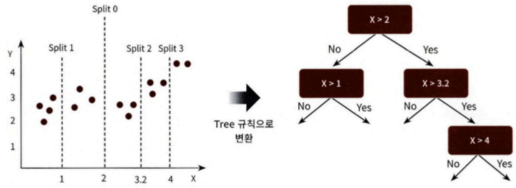
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score

# 데이터 로드
cancer = load_breast_cancer()

# 스케일링
scaler = StandardScaler()
data_scaled = scaler.fit_transform(cancer.data)

# 학습/테스트 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(
    data_scaled, cancer.target, test_size=0.3, random_state=0
)

# 로지스틱 회귀 모델 (기본 solver=lbfgs)
lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)

# 예측 및 평가
lr_preds = lr_clf.predict(X_test)
lr_preds_proba = lr_clf.predict_proba(X_test)[:, 1]

print('accuracy: {:.3f}, roc_auc: {:.3f}'.format(
    accuracy_score(y_test, lr_preds),
    roc_auc_score(y_test, lr_preds_proba)
))
```

5.8 회귀트리

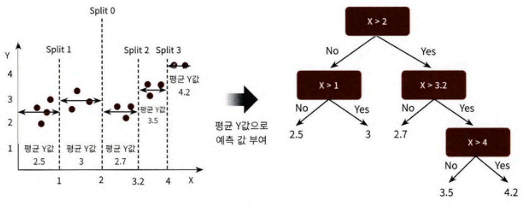
1. 개념 요약

- **회귀 트리(Regression Tree)** 는 선형 회귀처럼 회귀 계수를 기반으로 하지 않고,
  트리 구조(CART, Classification And Regression Tree) 를 이용해 예측값을 도출하는 모델

- 리프 노드에서 **데이터의 평균값을 예측값으로 사용**



리프 노드 생성 기준에 부합하는 트리 분할이 완료됐다면 리프 노드에 소속된 데이터 값의 평균값을 구해서 최종적으로 리프 노드에 결정 값으로 할당합니다.



- **분류 트리(Classification Tree)** 와 구조는 유사하나, 분류에서는 클래스 결정 / 회귀에서는 **값의 예측(평균)** 이 다름

2. CART 기반 회귀 모델

알고리즘	회귀용 Estimator	분류용 Estimator
Decision Tree	`DecisionTreeRegressor`	`DecisionTreeClassifier`
Random Forest	`RandomForestRegressor`	`RandomForestClassifier`
Gradient Boosting	`GradientBoostingRegressor`	`GradientBoostingClassifier`
XGBoost	`XGBRegressor`	`XGBClassifier`
LightGBM	`LGBMRegressor`	`LGBMClassifier`

3. 랜덤 포레스트를 이용한 보스턴 주택가격 예측

```
from sklearn.datasets import load_boston
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# 보스턴 데이터 로드
boston = load_boston()
bostonDF = pd.DataFrame(boston.data, columns=boston.feature_names)
bostonDF['PRICE'] = boston.target

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

# 랜덤포레스트 회귀 모델
rf = RandomForestRegressor(random_state=0, n_estimators=100)
neg_mse_scores = cross_val_score(rf, X_data, y_target,
```

```
                         scoring="neg_mean_squared_error", cv=5)

# RMSE 계산
rmse_scores = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print('5-Fold RMSE:', np.round(rmse_scores, 2))
print('평균 RMSE:', round(avg_rmse, 3))
```

```
5-Fold RMSE: [2.81 3.63 4.54 6.80 4.34]
평균 RMSE: 4.423
```