1.1 이론

1.1.0 머신러닝 파이프라인 개요

머신러닝 모델을 구축하고 적용하는 과정은 단순히 알고리즘을 선택하고 실행하는 것을 넘어, 다음과 같은 **전체 파이프라인 과정**으로 구성됩니다.

전체 흐름

```
데이터 수집 → 데이터 전처리 → 특성 선택 → 모델 선택 → 학습 → 평가 → (재)튜닝 → 예측/활용
```

각 단계 설명

1. 데이터 수집

- CSV, Excel, 데이터베이스, 센서 등 다양한 소스에서 데이터 수집
- 예: Kaggle의 부동산 데이터, 공공 데이터 포털

2. 데이터 전처리

- 결측치 처리, 이상치 제거, 범주형 인코딩, 정규화 등
- 예: `StandardScaler`, `OneHotEncoder`, `SimpleImputer`

3. 특성 선택 및 차원 축소

- 모델 성능에 영향을 주는 핵심 변수 선별
- PCA 같은 차원 축소 기법 사용

4. 모델 선택 및 학습

- 선형 회귀, 의사결정트리, 로지스틱 회귀 등 문제에 맞는 모델 선택
- 훈련 데이터를 통해 모델 학습

5. 성능 평가

- 테스트셋을 사용하여 예측 성능 평가
- 회귀: MSE, MAE, R^2 / 분류: 정확도, F1, ROC-AUC

6. 모델 튜닝

- 하이퍼파라미터 조정, 특성 재선택 등으로 성능 향상
- `GridSearchCV`, `RandomizedSearchCV` 활용

7. 예측 및 활용

- 실제 문제에 적용 (미래 값 예측, 분류 자동화 등)

1.1.1 회귀 분석이란?

회귀 분석은 주어진 특성(입력 변수)을 바탕으로 **수치적인 연속값을 예측**하는 지도학습(Supervised Learning)의 대표적인 방법입니다.

예시:

- 집의 면적과 방 수로 가격 예측
- 수면 시간과 학습 시간으로 수능 점수 예측
- 환경 데이터로 탄소 배출량 예측

지도학습의 두 형태 정리:

구분	목표값	예시	대표 알고리즘
회귀	연속값 (수치형)	가격, 점수, 시간	선형 회귀, 다항 회귀 등
분류	범주 (class label)	합/불합, 고양이/개	로지스틱 회귀, SVM 등

1.1.2 선형 회귀 모델의 수식 구조

선형 회귀(Linear Regression)는 가장 기본적인 회귀 모델로, 입력 변수들과 출력 변수 사이에 **선형 관계**가 있다고 가정합니다.

가설 함수 수식:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

여기서 θ_j 는 모델이 학습해야 할 계수이며, x_j 각 특성(feature)입니다.

1.1.3 비용 함수 (Loss Function)

모델이 얼마나 잘 예측했는지를 측정하기 위해 가장 대표적인 손실 함수로 평균 제곱 오차(MSE)를 사용합니다.

비용 함수 (MSE):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

m: 전체 학습 데이터의 개수

$x^{(i)}$: i 번째 입력 샘플

$y^{(i)}$: i 번째 실제 정답값

$h_\theta(x^{(i)})$: i 번째 예측값

$J(\theta)$: 파라미터 θ에 대한 비용 함수

1.1.4 경사 하강법 (Gradient Descent)

오차를 줄이기 위해 θ를 반복적으로 업데이트합니다. 미분을 통해 오차의 기울기를 계산하고 그 반대 방향으로 파라미터를 반복적으로 갱신합니다.

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

θ_j: j번째 파라미터 (가중치)

α: 학습률 (learning rate)

$\frac{\partial J}{\partial \theta_j}$: θ_j에 대한 비용 함수의 미분값

1.1.5 다항 회귀 (Polynomial Regression)

입력 x를 다항식으로 확장해 더 복잡한 비선형 관계를 표현할 수 있습니다. 선형회귀는 입력과 출력 간 관계가 직선일때만 잘 작동합니다. 다항식으로 확장이 되면 곡선 형태의 데이터에 대해서도 잘 작동할 수 있습니다.

$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots + \theta_d x^d$$

x: 입력 변수

d: 다항식 차수 (degree)

θ_i: 다항 항(i차 항)의 계수

1.1.6 정규화 회귀 (Regularized Regression)

다항 회귀나 고차원 데이터에서는 모델이 훈련 데이터에만 너무 잘 맞는 **과적합(overfitting)** 문제가 자주 발생합니다. 이를 해결하기 위해 비용 함수에 **패널티 항**을 추가하는 방식이 정규화입니다.

(1) Ridge 회귀 (L2 정규화)

> ridge : 산등성이, 산마루

오차를 최소화하면서 동시에 파라미터들이 제곱합이 너무 커지지 않도록 제한합니다.

$$J_{ridge}(\theta) = J(\theta) + \lambda \sum_{j=1}^{n} \theta_j^2$$

λ: 정규화 강도 (규제 계수) : 값이 클 수록 제약이 강해짐

$\sum_{j=1}^{n} \theta_j^2$: 파라미터 제곱합 (L2 norm) : 파라미터를 0에 가깝게 만들지만, 완전히 0으로 만들진 않음.

(2) Lasso 회귀 (L1 정규화)

> LASSO : Least Absolute Shrinkage and Selection Operator

오차를 최소화하면서 파라미터 절댓값의 합이 너무 커지지 않도록 제한합니다.

$$J_{lasso}(\theta) = J(\theta) + \lambda \sum_{j=1}^{n} |\theta_j|$$

$\sum_{j=1}^{n} |\theta_j|$: 파라미터 절댓값의 합 (L1 norm)

일부 파라미터가 완전히 0이 되기 때문에, 불필요한 특성을 제거하는 효과가 있음.

고차원에서의 변수 선택 역할 수행 가능

1.1.7 회귀 모델 비교 요약

모델	과적합 제어	특성 선택	해석 용이성	표현력
선형 회귀	×	×	높음	낮음
다항 회귀	×	×	중간	높음
릿지 회귀	O (L2)	×	중간	중간
라쏘 회귀	O (L1)	O	중간	중간

- **과적합 제어**: 모델이 훈련 데이터에만 과도하게 맞춰지는 현상(과적합)을 방지하기 위해 정규화 항(릿지:L2, 라쏘:L1)을 사용하여 파라미터 크기를 제한하는 방법입니다.
- **특성 선택**: 예측에 중요하지 않은 특성(feature)의 가중치를 0으로 만들어 유용한 특성만 자동으로 선택하는 방법이며 라쏘(L1) 회귀에서 수행됩니다.
- **해석 용이성**: 모델이 어떤 특성이 결과에 얼마나 영향을 미쳤는지 명확한 수치(계수 등)로 제시하여 사용자가 예측 과정과 이유를 직관적으로 이해할 수 있는 정도를 말합니다.
- **표현력**: 데이터가 가진 복잡한 패턴과 비선형적 관계를 모델이 얼마나 유연하고 정확하게 표현할 수 있는지를 나타내는 모델의 능력을 의미합니다.

1.2 수식

1.2.1 평균 제곱 오차(MSE) 도출

회귀 문제에서 모델이 얼마나 잘 예측했는지를 수치로 평가하려면, 예측값과 실제값의 차이를 계산해야 합니다.

이 차이를 바탕으로 정의되는 가장 널리 쓰이는 손실 함수는 **평균 제곱 오차 (Mean Squared Error)** 입니다.

1.2.1.1 목적: 오차의 제곱을 평균 내자

각 샘플 $x^{(i)}$ 에 대해 예측값은 $h_\theta(x^{(i)}) = \theta^T x^{(i)}$, 실제값은 $y^{(i)}$입니다.

따라서 예측 오차는 $\theta^T x^{(i)} - y^{(i)}$이며, 이를 제곱해서 더한 뒤 평균을 내면 전체 오차를 대표할 수 있습니다.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

1.2.2 경사 하강법을 위한 미분 유도

모델의 학습이란 결국 이 비용 함수 $J(\theta)$를 최소화하는 θ를 찾는 것입니다.

이를 위해 우리는 **경사 하강법 (Gradient Descent)** 알고리즘을 적용합니다.

그 핵심은 각 파라미터 θ_j에 대해 **기울기(gradient)** 를 구하는 것입니다.

1.2.2.1 비용 함수에 대한 편미분

θ_j대해 편미분하면:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)}) \cdot x_j^{(i)}$$

1.2.2.2 경사 하강법 업데이트 식 유도

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta_j}$$

즉, 각 파라미터는 그에 대한 비용 함수의 기울기만큼, **오차가 줄어드는 방향으로** 이동합니다.

> 학습률 α 는 너무 크면 발산, 너무 작으면 수렴이 느려서, 적절한 값 설정이 중요합니다.

1.2.3 전체 수식 요약

비용함수

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$$

기울기

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)}) x_j^{(i)}$$

경사 하강법 업데이트 식

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta_j}$$

1.2.4 정규화 회귀 – 미분 관점에서의 수식 유도

머신러닝에서 모델의 복잡도가 지나치게 커져 학습 데이터에 과적합(overfitting)되는 것을 막기 위해 **정규화(regularization)** 를 도입합니다.

정규화는 손실 함수에 **패널티 항**을 추가하여 모델이 너무 큰 계수를 갖지 못하게 억제합니다.

1.2.4.1 Ridge 회귀 (L2 정규화)

Ridge 회귀는 제곱합을 기반으로 정규화하는 방법입니다.

비용 함수

$$J_{\text{ridge}}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\theta^T x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

편미분 (Gradient)

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

- 기존의 MSE 오차 기울기에 **정규화 항의 기울기** $\frac{\lambda}{m}\theta_j$ 가 추가됩니다.
- 이 항은 θ_j의 절대 크기에 비례하여, θ_j를 **0 방향으로 부드럽게 수축**시킵니다.
- Ridge는 **모든 파라미터를 조금씩 줄이며**, 과도하게 커진 계수를 억제하는 방식입니다.
- 하지만 $\theta_j = 0$이 되도록 강제하지는 않기 때문에 **희소성(sparsity)은** 유도하지 않습니다.

1.2.4.2 Lasso 회귀 (L1 정규화)

Lasso 정규화는 계수의 **절댓값 합**을 패널티로 더함으로써,

일부 계수가 **완전히 0이 되도록** 만들어 **특성 선택(feature selection)** 효과를 냅니다.

비용 함수

$$J_{textlasso}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2 + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j|$$

편미분 (서브그라디언트)

L1 정규화는 절댓값이 포함되어 있어 $\theta_j = 0$에서 미분이 정의되지 않으므로,

서브그라디언트(subgradient) 를 사용합니다.

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\cdot \text{sign}(\theta_j)$$

- $sign(\theta_j)$는 $\theta_j > 0$일 때 $+1$, $\theta_j < 0$일 때 -1, $\theta_j = 0$일 땐 $[-1, +1]$ 범위입니다.
- 이 항은 일정한 크기의 상수 힘으로 θ_j를 **0 방향으로 끌어당기며**, θ_j가 작을수록 쉽게 0이 되도록 유도합니다.
- 결과적으로 Lasso는 일부 계수를 완전히 0으로 만들어, **희소한 모델(sparse model)** 을 형성합니다.

1.2.5 정규화 회귀 – 기하학적 관점에서의 해석

정규화 회귀는 **최적화 문제에 제약 조건을 추가한 형태**로 이해할 수 있으며, 이를 통해 Ridge와 Lasso의 동작 차이를 **기하학적으로 직관적**으로 이해할 수 있습니다.

1.2.5.1 제약 조건 최적화 형태로의 변환

Ridge나 Lasso에서 손실 함수에 정규화 항을 더하는 방식은, 아래와 같이 **제약 조건을 둔 최적화 문제**와 동치입니다.

> $L(\theta)$가 가장 작아지도록 하는 파라미터 θ 를 찾아라. 동시에 $R(\theta) \le t$ 라는 제약 조건을 만족해야 한다.

$$\min_{\theta} \; L(\theta) \quad \text{subject to} \quad R(\theta) \leq t$$

- $L(\theta)$: 손실 함수 (예: 평균제곱오차)
- $R(\theta)$: 정규화 항 (규제 함수) $\|\theta\|_2^2$, $\|\theta\|_1$
- t: 제약의 강도를 조절하는 상수, 허용 가능한 복잡도의 상한선
- 해를 찾을 때, 반드시 $R(\theta) \leq t$ 범위 안에서만 θ를 탐색합니다.

Ridge의 제약 영역

$$R(\theta) = \sum_{j=1}^{n} \theta_j^2 \quad \Rightarrow \quad \text{제약 영역은 원 또는 구의 형태}$$

Lasso의 제약 영역

$$R(\theta) = \sum_{j=1}^{n} |\theta_j| \quad \Rightarrow \quad \text{제약 영역은 마름모 또는 정육면체 형태}$$

1.2.5.2 손실 함수 등고선과 제약 영역의 교점

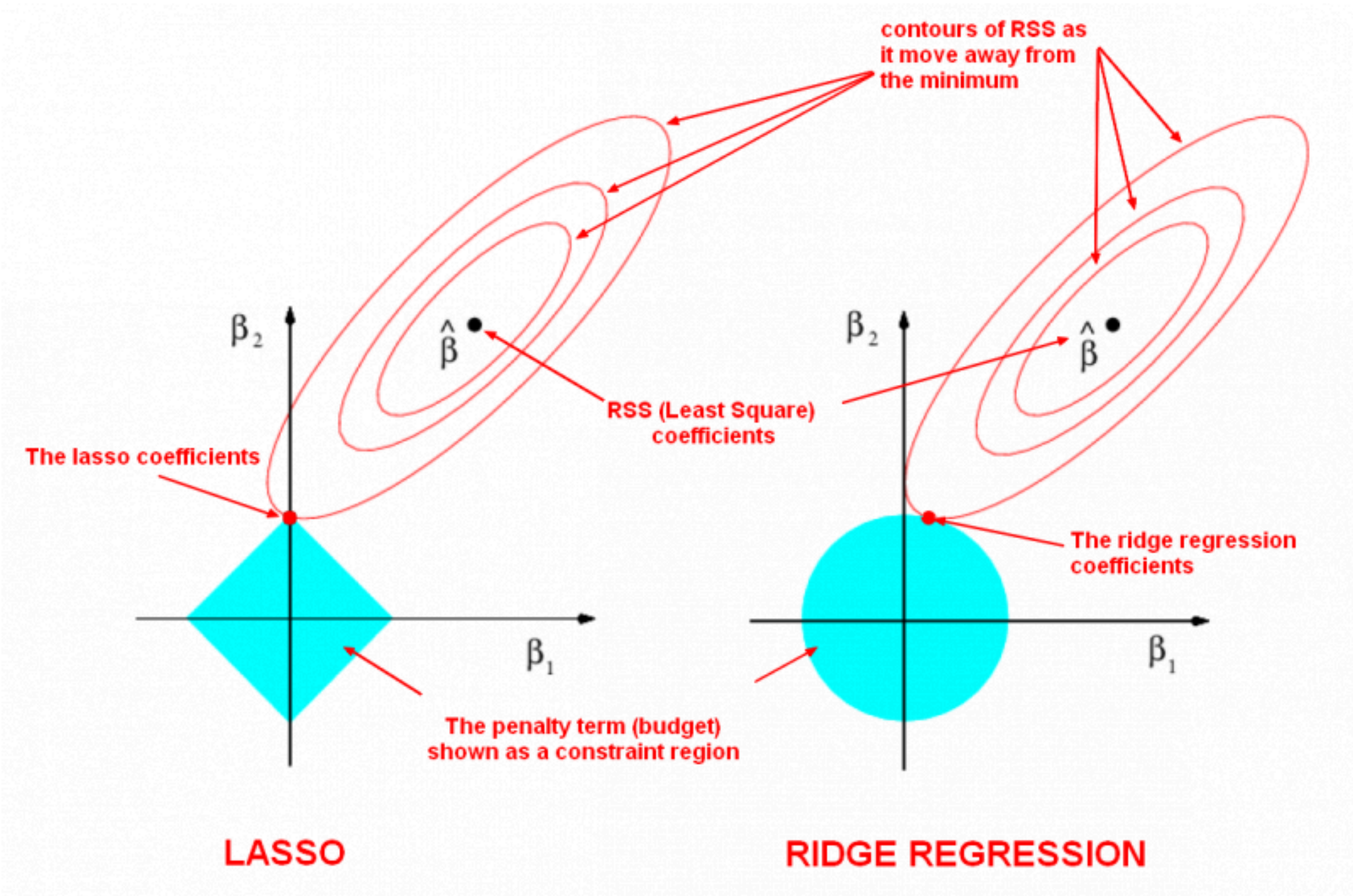
손실 함수 $L(\theta)$는 일반적으로 **타원 형태의 등고선**을 갖습니다. 최적해는 이 등고선이 **제약 영역의 경계**와 처음 **접하는 지점**에서 결정됩니다.

Ridge의 경우

- 원형 제약 영역은 매끄럽기 때문에 등고선이 일반적으로 중심 근처 **곡면**과 접함
- 결과적으로 대부분의 θ_j는 **작지만 0이 아님**

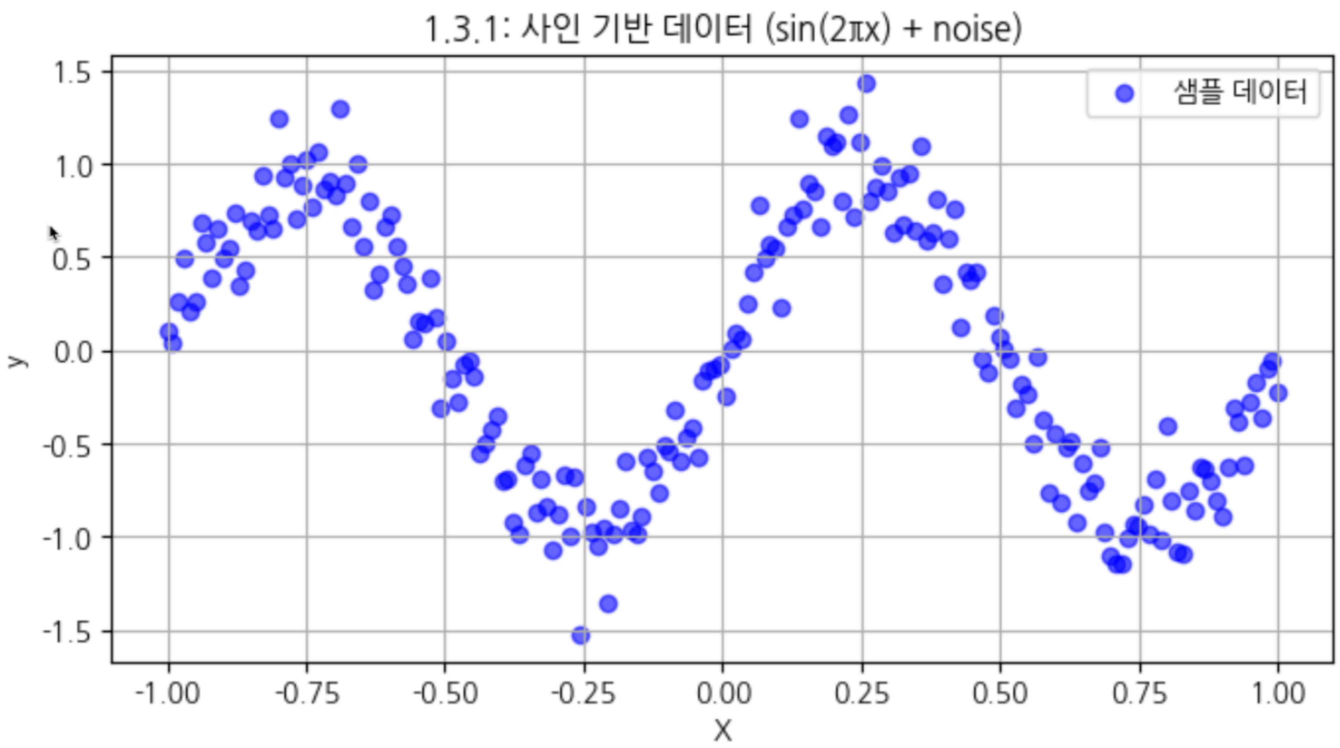
Lasso의 경우

- 마름모 제약 영역은 **꼭짓점이 축 방향 위에 있음**
- 등고선이 꼭짓점과 자주 접하게 되어, 일부 $\theta_j = 0$이 되는 **희소 해(sparse solution)** 가 도출됨



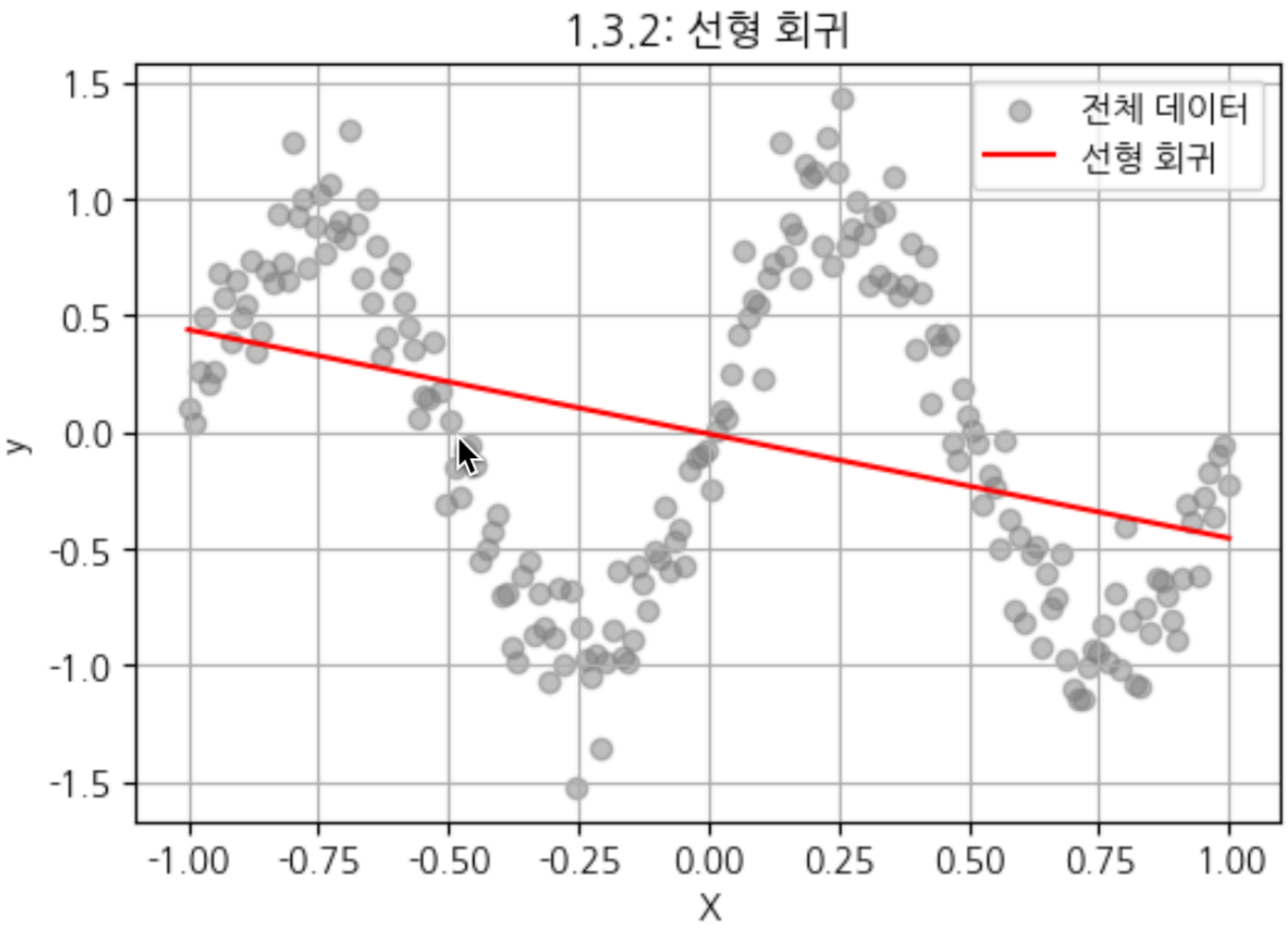
1.3 코드

```
# 1.3.1: 사인 함수 기반 고차 다항 데이터 생성
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

np.random.seed(42)
# 입력은 [-1,1] 균등 분포
X = np.linspace(-1, 1, 200).reshape(-1, 1)
# 실제 함수는 sin(2πx), 여기에 약간의 노이즈 추가
y = np.sin(2 * np.pi * X).ravel() + 0.2 * np.random.randn(200)

# (훈련/테스트 분할 사용하지 않음)
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

plt.figure(figsize=(8, 4))
plt.scatter(X, y, color='blue', alpha=0.6, label='샘플 데이터')
plt.title("1.3.1: 사인 기반 데이터 (sin(2πx) + noise)")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```

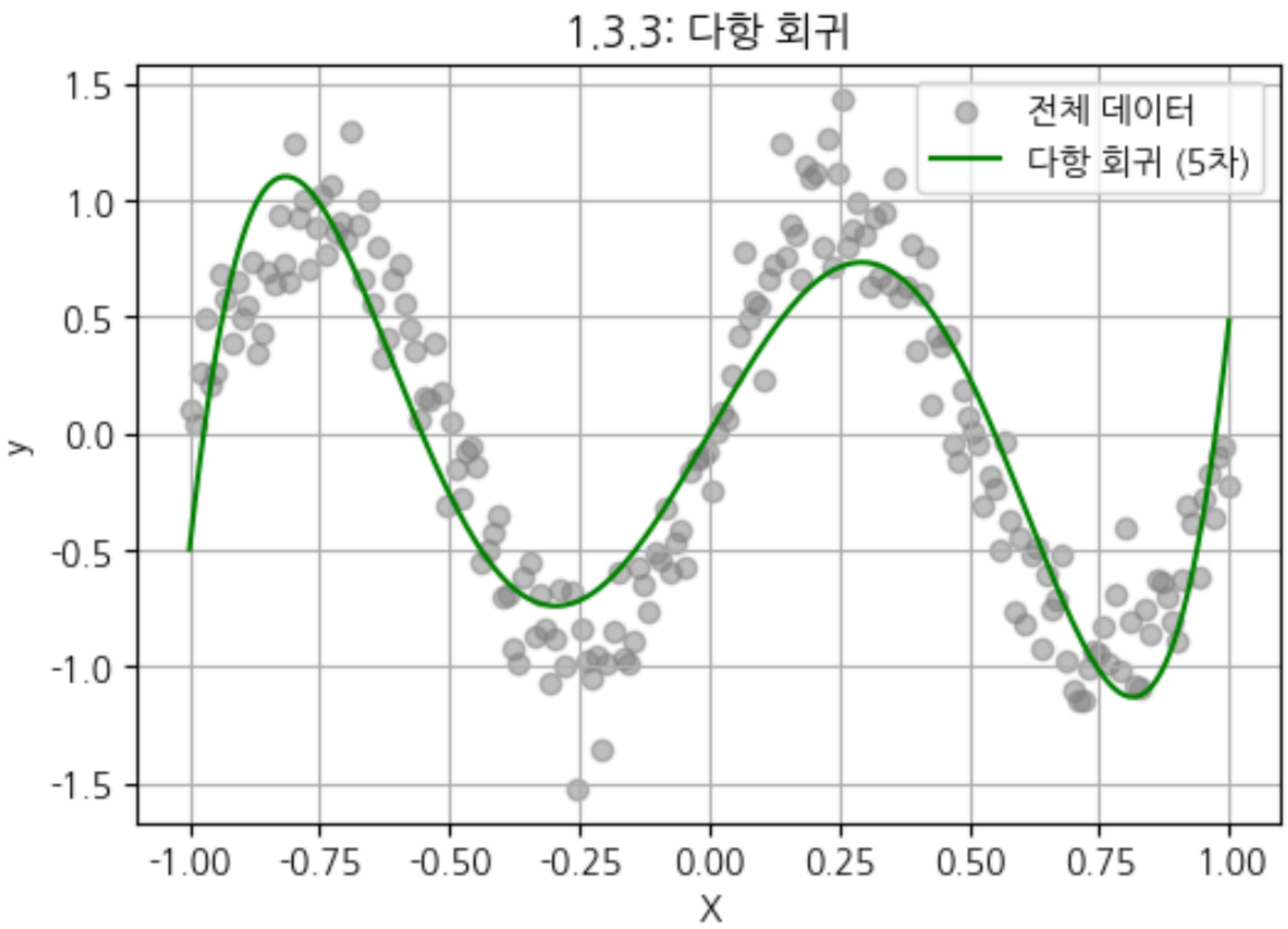



```
# 1.3.2: 선형 회귀
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)

# 정렬된 X 값에 대해 예측
X_sorted = np.sort(X, axis=0)
y_lin = lin_reg.predict(X_sorted)

# 시각화
plt.figure(figsize=(6, 4))
plt.scatter(X, y, color='gray', alpha=0.5, label='전체 데이터')
```

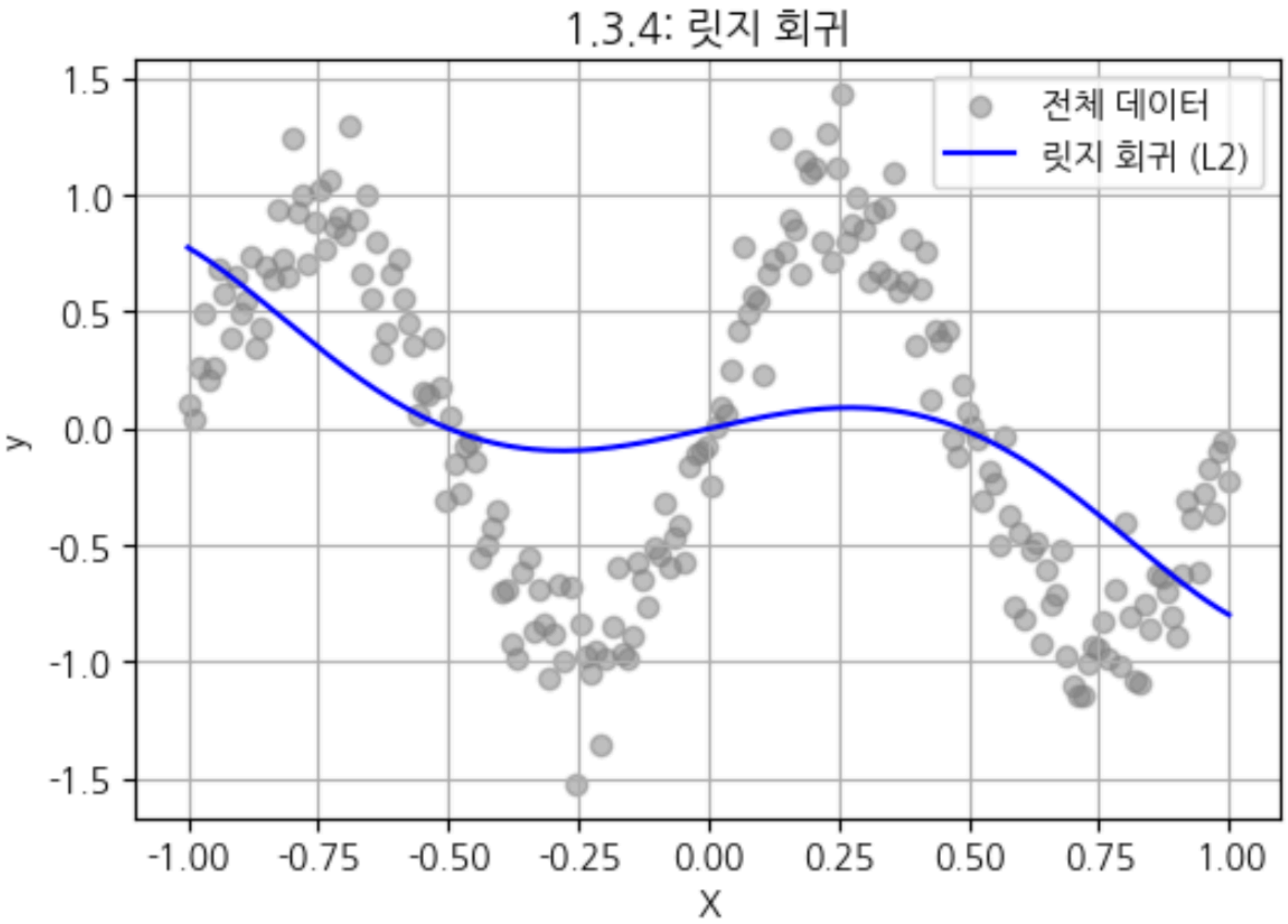
```
plt.plot(X_sorted, y_lin, color='red', label='선형 회귀')
plt.title("1.3.2: 선형 회귀")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```



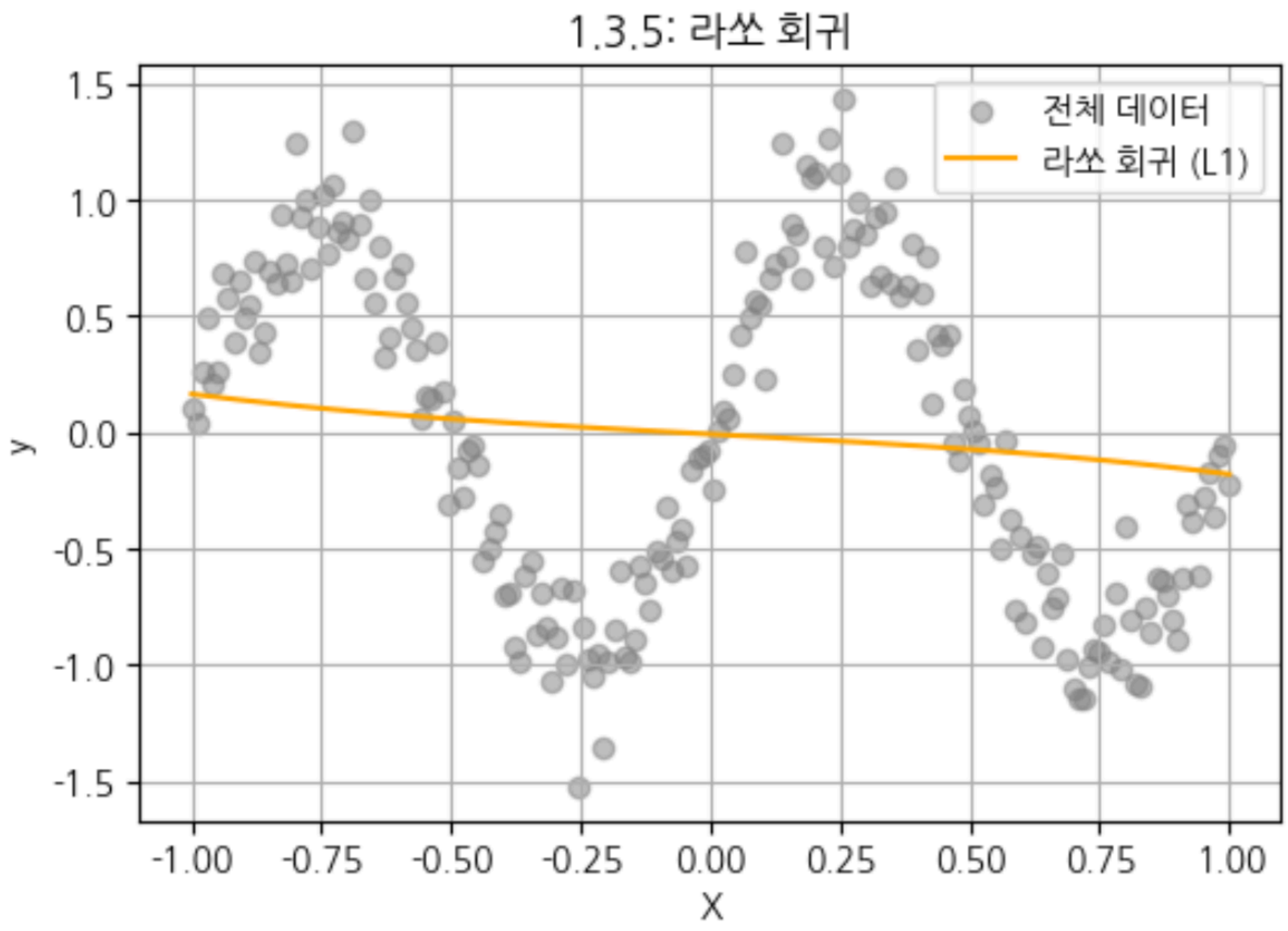

```
# 1.3.3: 다항 회귀 (5차)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=5)
X_poly = poly.fit_transform(X)
X_sorted_poly = poly.transform(X_sorted)

lin_poly = LinearRegression()
lin_poly.fit(X_poly, y)
y_poly = lin_poly.predict(X_sorted_poly)

# 시각화
plt.figure(figsize=(6, 4))
plt.scatter(X, y, color='gray', alpha=0.5, label='전체 데이터')
plt.plot(X_sorted, y_poly, color='green', label='다항 회귀 (5차)')
plt.title("1.3.3: 다항 회귀")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```



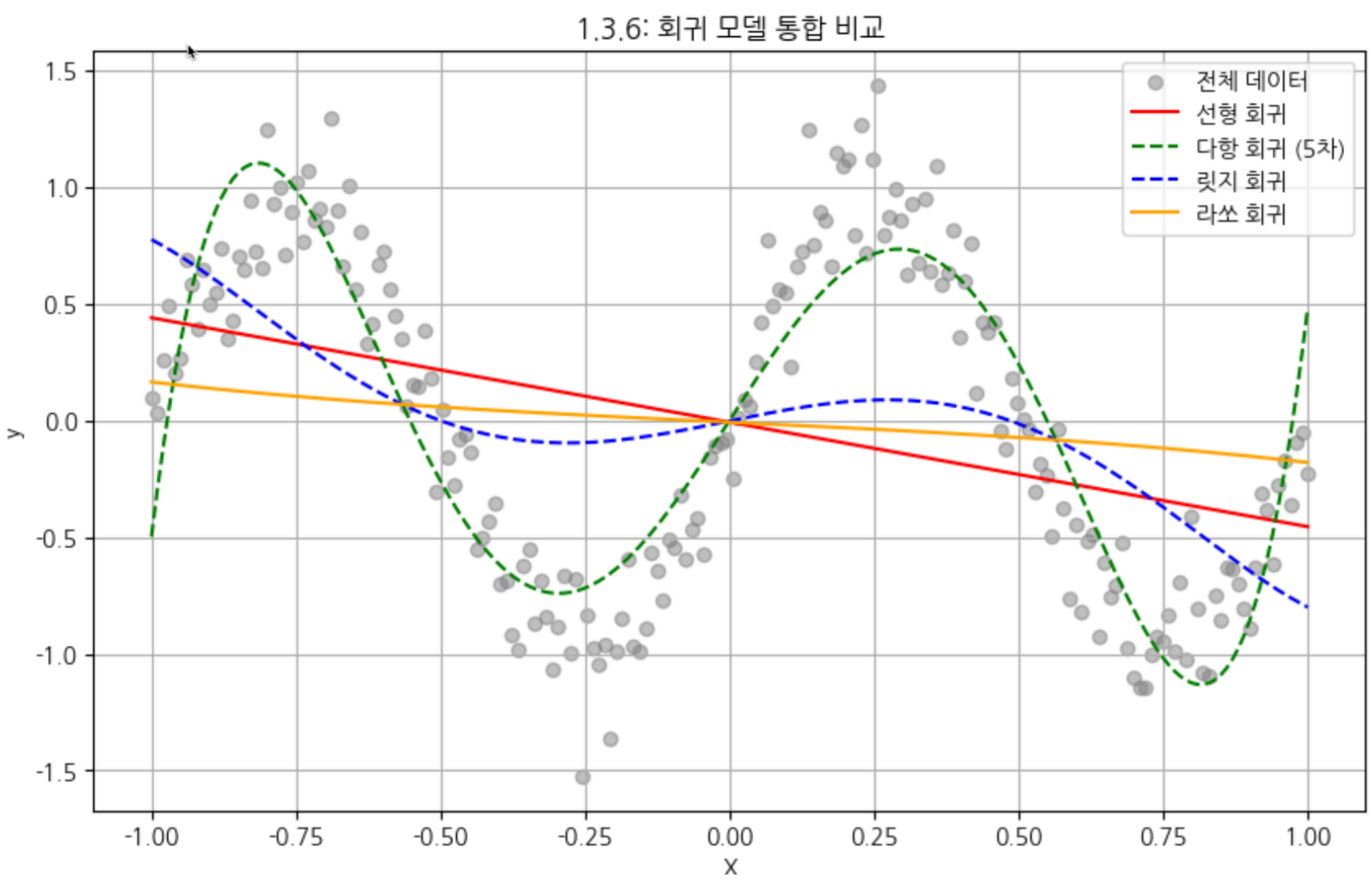

```
# 1.3.4: 릿지 회귀
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_poly, y)
y_ridge = ridge.predict(X_sorted_poly)

# 시각화
plt.figure(figsize=(6, 4))
plt.scatter(X, y, color='gray', alpha=0.5, label='전체 데이터')
plt.plot(X_sorted, y_ridge, color='blue', label='릿지 회귀 (L2)')
plt.title("1.3.4: 릿지 회귀")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```




```
# 1.3.5: 라쏘 회귀
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1, max_iter=10000)
lasso.fit(X_poly, y)
y_lasso = lasso.predict(X_sorted_poly)

# 시각화
plt.figure(figsize=(6, 4))
plt.scatter(X, y, color='gray', alpha=0.5, label='전체 데이터')
plt.plot(X_sorted, y_lasso, color='orange', label='라쏘 회귀 (L1)')
plt.title("1.3.5: 라쏘 회귀")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```




```
# 1.3.6: 네 가지 회귀 모델 통합 비교
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='gray', alpha=0.5, label='전체 데이터')
plt.plot(X_sorted, y_lin, color='red', label='선형 회귀')
plt.plot(X_sorted, y_poly, color='green', linestyle='--', label='다항 회귀 (5차)')
plt.plot(X_sorted, y_ridge, color='blue', linestyle='--', label='릿지 회귀')
plt.plot(X_sorted, y_lasso, color='orange', linestyle='-', label='라쏘 회귀')
plt.title("1.3.6: 회귀 모델 통합 비교")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```



1.4 결과 해석

1.4.1 MSE와 R² 계

회귀 모델의 성능을 평가할 때 대표적으로 사용되는 지표는 다음 두 가지입니다.

(1) 평균 제곱 오차 (Mean Squared Error, MSE)

MSE는 실제값과 예측값 사이의 **제곱 차이의 평균**입니다. 작을수록 예측이 실제와 가깝다는 뜻입니다.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

(2) 결정 계수 (R^2 Score)

R^2 는 예측값이 실제값을 얼마나 잘 설명하는지를 나타냅니다. **1에 가까울수록 좋은 성능**입니다.

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y})^2}p$$

1.4.2 평가 코드

```
#1.4.2

from sklearn.metrics import mean_squared_error, r2_score

# 네 가지 회귀 모델 평가
models = {
    "선형 회귀": (lin_reg, X),
    "다항 회귀": (lin_poly, X_poly),
    "릿지 회귀": (ridge, X_poly),
    "라쏘 회귀": (lasso, X_poly),
}
print("모델\t\t\tMSE\t\t\tR²")
print("-" * 50)
for name, (model, X_input) in models.items():
    y_pred = model.predict(X_input)
    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)
    print(f"{name:<10s}\t{mse:.4f}\t\t{r2:.4f}")
```

모델 MSE R²

선형 회귀 0.6133 0.0022
다항 회귀 0.5405 0.1207
릿지 회귀 0.5433 0.1162
라쏘 회귀 0.5776 0.0603

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import learning_curve
```

```
# 1. 데이터 생성 (복잡도  + 잡음 )
np.random.seed(42)
X = np.linspace(-1.5, 1.5, 200).reshape(-1, 1)
y = (
    0.3 * np.sin(9 * np.pi * X).ravel()   # 고주파수로 변경
    + 0.3 * X.ravel()**5                  # 곡선성
    + 0.6 * np.random.randn(200)          # 잡음
)

# 2. 모델 정의 (15차 다항)
lin_reg = LinearRegression()
poly15_lin = make_pipeline(PolynomialFeatures(degree=15), LinearRegression())
poly15_ridge = make_pipeline(
    PolynomialFeatures(degree=15),
    StandardScaler(),
    Ridge(alpha=0.005)
)

poly15_lasso = make_pipeline(
    PolynomialFeatures(degree=15),
    StandardScaler(),
    Lasso(alpha=0.001, max_iter=100000)
)

models = {
    "선형 회귀 (1차)": (lin_reg, X),
    "다항 회귀 (15차)": (poly15_lin, X),
    "릿지 회귀 (15차, α=0.005)": (poly15_ridge, X),
    "라쏘 회귀 (15차, α=0.001)": (poly15_lasso, X),
}

# 3. 학습 곡선 시각화
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.ravel()

for idx, (name, (model, X_input)) in enumerate(models.items()):
    train_sizes, train_scores, val_scores = learning_curve(
        model, X_input, y,
        train_sizes=np.linspace(0.1, 1.0, 10),
        scoring='neg_mean_squared_error',
        cv=5,
        shuffle=True,
        random_state=42
    )

    # 음수 제거 + log 스케일 호환성 확보
    epsilon = 1e-6
    train_errors = np.clip(-np.mean(train_scores, axis=1), epsilon, None)
    val_errors = np.clip(-np.mean(val_scores, axis=1), epsilon, None)

    ax = axes[idx]
    ax.plot(train_sizes, train_errors, 'o-', label="훈련 MSE", color='orange', linewidth=2)
    ax.plot(train_sizes, val_errors, 'o--', label="검증 MSE", color='orangered', linewidth=2)
    ax.set_title(f"{name}")
    ax.set_xlabel("훈련 샘플 수")
    ax.set_ylabel("평균 제곱 오차 (MSE)")
```

```
    # 조건부 로그 스케일 적용
    if "선형 회귀" not in name:
        ax.set_yscale("log")

    ax.legend()
    ax.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.tight_layout()
plt.savefig("learning_curve_poly15_mixed_scale.png")
plt.show()
```