



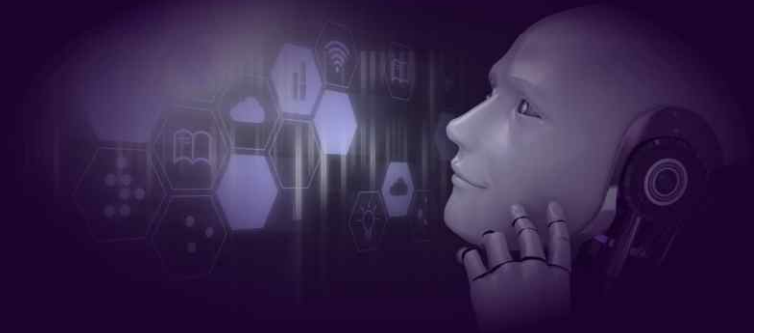
영진전문대학교

글로벌시스템융합과 - GSC

Supervised Learning: Regression

정영철 교수

글로벌시스템융합과



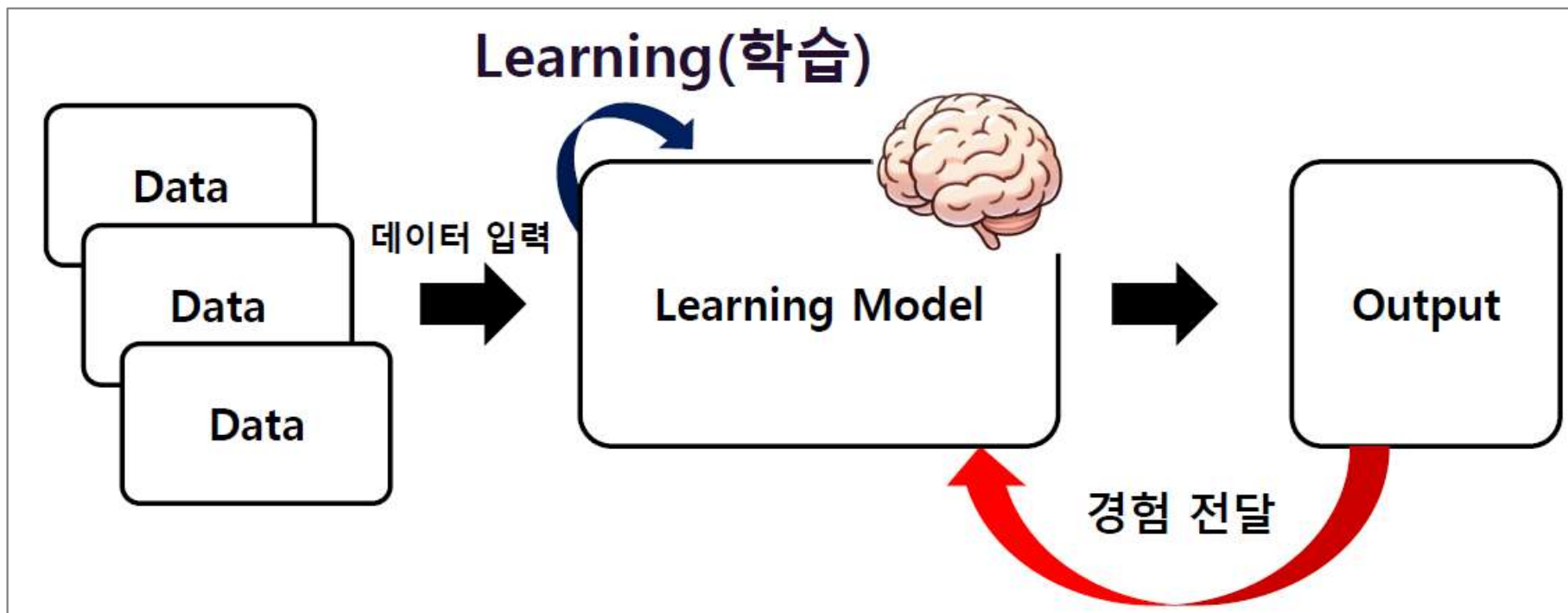
A.I WITH BETTER LIFE

ML(Machine Learning) [기계학습] 이란?

컴퓨터가 명시적으로 프로그래밍 되지 않아도

데이터로부터 학습하여 스스로 패턴을 찾고

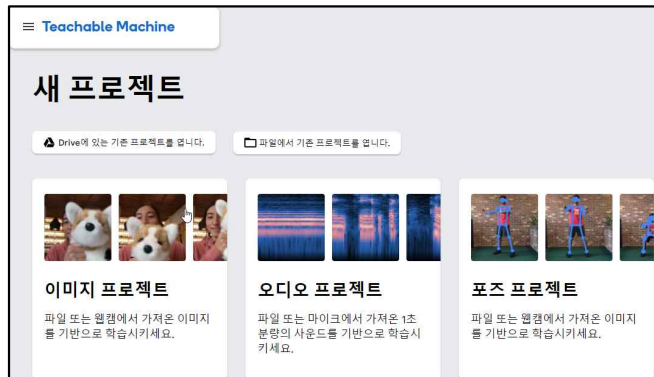
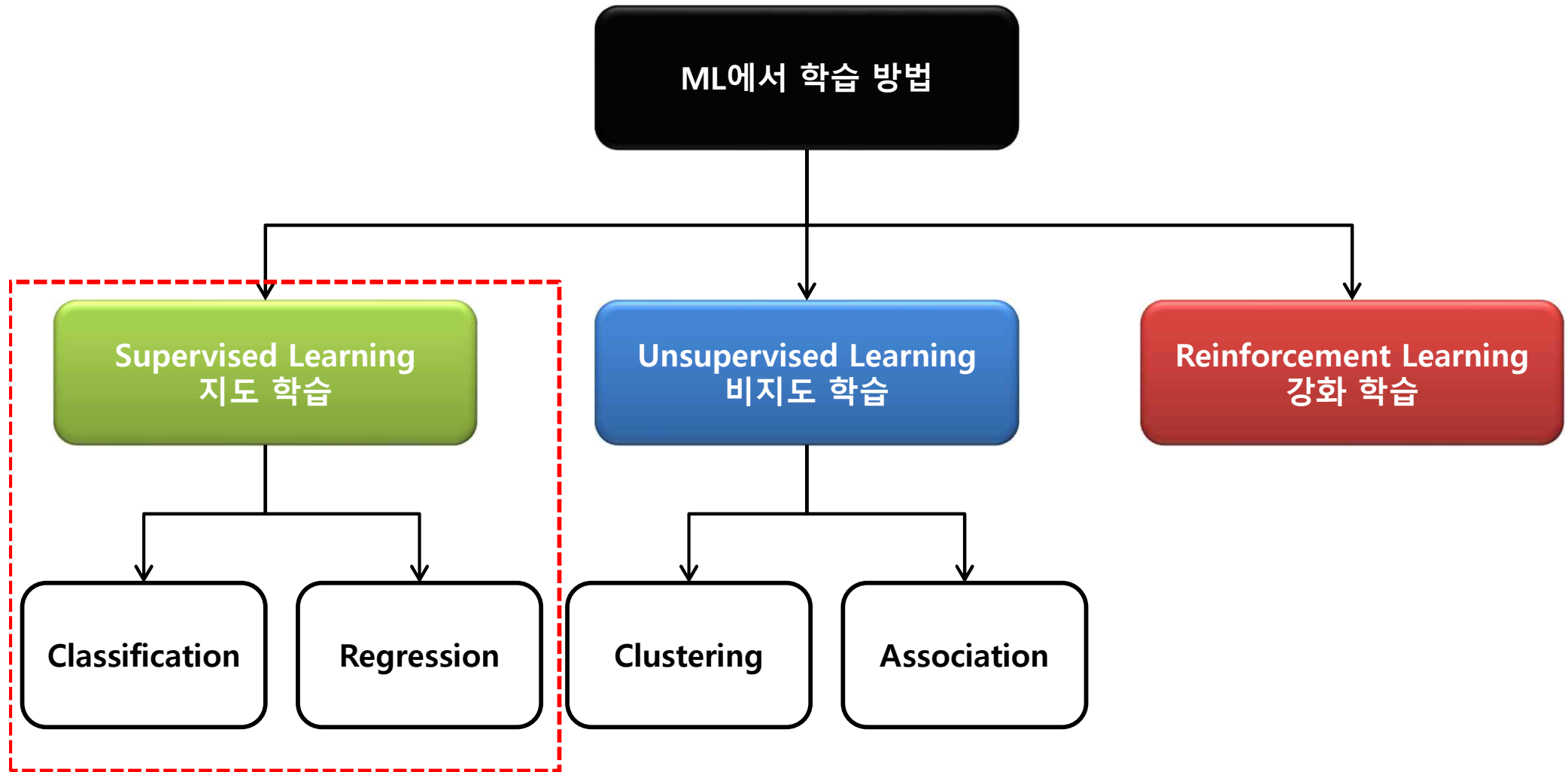
의사 결정을 내리는 기술



ML(머신러닝) 학습 과정: 데이터 → 모델 훈련 → 예측 → 평가)

- 데이터 기반 학습: 모델이 대량의 데이터를 분석하여 패턴을 학습
- 일반화(Generalization): 학습한 패턴을 새로운 데이터에도 적용할 수 있도록 함
- 알고리즘: 데이터를 학습하고 결과를 예측하는 다양한 수학적/통계적 기법

Machine Learning에서 학습 방법 (1)



기계 학습(ML)의 세 가지 학습 방법

- 기계 학습(ML)에서는 세 가지 주요 학습 방법이 사용 된다.

**Supervised
Learning**

지도 학습

Label(정답) 필요

**Unsupervised
Learning**

비지도 학습

Label(정답) 불필요

**Reinforcement
Learning**

강화 학습

보상(Reward) 시스템

지도 학습(Supervised Learning):

- 정답(라벨)이 주어진 데이터를 사용하여, 컴퓨터가 입력과 출력 간의 관계를 학습하는 방식
- 컴퓨터는 제공된 입력을 기반으로 정답을 예측하는 규칙을 학습한다

비지도 학습(Unsupervised Learning):

- 라벨이 주어지지 않은 데이터를 이용하여, 컴퓨터가 스스로 숨겨진 패턴이나 구조를 찾아내는 학습 방식.

강화 학습(Reinforcement Learning):

- 컴퓨터(에이전트)가 시행착오를 거치면서 최적의 행동을 학습하는 방식이다.
- 이 과정에서 행동의 결과에 따라 보상(예: 게임에서 점수 획득)을 받아, 보상을 극대화하는 방향으로 학습한다.

지도 학습 (Supervised Learning)의 종류: 문제 유형

Supervised Learning (지도학습)

- 분류: 데이터가 특정 카테고리(클래스)로 분류

- 회귀: 연속적인 숫자(실수)를 예측하는 방법

Classification (분류)

- Logistic Regression
- Naive Bayes
- K-Nearest Neighbors
- Decision Tree
- Support Vector Machines

Binary Classification
(이진분류)

Multiple Classification
(다중 분류)

Regression (회귀)

- Linear Regression
- Ridge Regression (L2 Regularization)
- Lasso Regression (L1 Regularization)
- Neural Networks Regression
- Decision Tree

오늘의 주제

← Algorithms →

- 회귀(Regression)는 독립 변수(입력 변수)와 종속 변수(출력 변수) 간의 관계를 수학적 모델로 표현하는 방법
- 분류(Classification)는 독립 변수(입력 변수)를 입력 받아 미리 정의된 범주(Category)의 요소를 선택하는 것 (이산)

Legacy ML(Machine Learning) algorithms vs DL (Deep Learning)

① 대부분의 문제에서 DL이 유리한 것은 맞음

- 이미지 처리(CNN), 자연어 처리(Transformer), 자율주행, 음성 인식 등에서는 딥러닝이 기존 방법보다 압도적인 성능을 보임
- 특히 대규모 데이터(Big Data)와 고성능 하드웨어(GPU, TPU)가 있는 경우 DL이 거의 모든 분야에서 최강

② 하지만 기존 머신러닝 알고리즘도 여전히 많이 사용됨

- 딥러닝이 항상 최적의 선택은 아님
- 데이터가 작거나, 연산 비용이 적은 모델이 필요한 경우에는 기존 머신러닝 알고리즘이 훨씬 효율적임

③ 딥러닝은 많은 데이터가 필요하고 계산 비용이 높음

- 데이터가 적다면 DL은 과적합(Overfitting)이 심하게 발생할 수 있음
- 기업에서 실무 적용 시, GPU 비용이 높기 때문에 가벼운 모델이 더 선호되는 경우가 많음

④ 머신러닝과 딥러닝을 같이 쓰는 경우도 많음 (Hybrid Approach)

- 일부 문제에서는 딥러닝으로 특징(Feature)를 추출하고, 머신러닝 모델로 최종 예측을 수행하는 방식이 더 좋은 결과를 냄

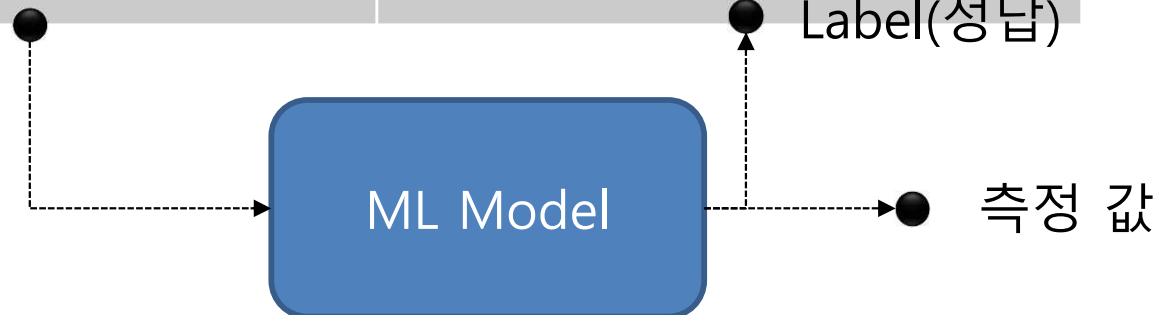
비교 항목	딥러닝 (DL)	기존 머신러닝 (ML)
데이터 필요량	• 많아야 함	• 적은 데이터도 가능
연산 비용	• GPU 필요, 고비용	• CPU만으로도 충분
해석 가능성	• 복잡해서 어려움 (Black Box)	• 모델이 직관적이고 설명 가능
적용 분야	• 이미지, 음성, 텍스트	• 수치 데이터, 표 데이터(Excel, 금융)
과적합 위험	• 데이터가 적으면 과적합 발생	• 적은 데이터에서도 안정적

Linear Regression 예제

- ✓ 공부 시간과 성적 간의 관계를 학습하는 모델을 생성하고, 특정 공부 시간을 입력하면 예상 성적을 예측하는 머신러닝(ML) 시스템을 개발한다

- Training Data Set (학습데이터 셋)

X(hour)	Y(score)
10	90
9	80
3	50
2	30
1	10



Regression Modeling 절차 (1)

- 학습 데이터셋 (Training Data Set) :
 - 머신러닝(ML) 모델을 학습하는 데 사용되는 데이터

- Training Data Set (학습 데이터셋)

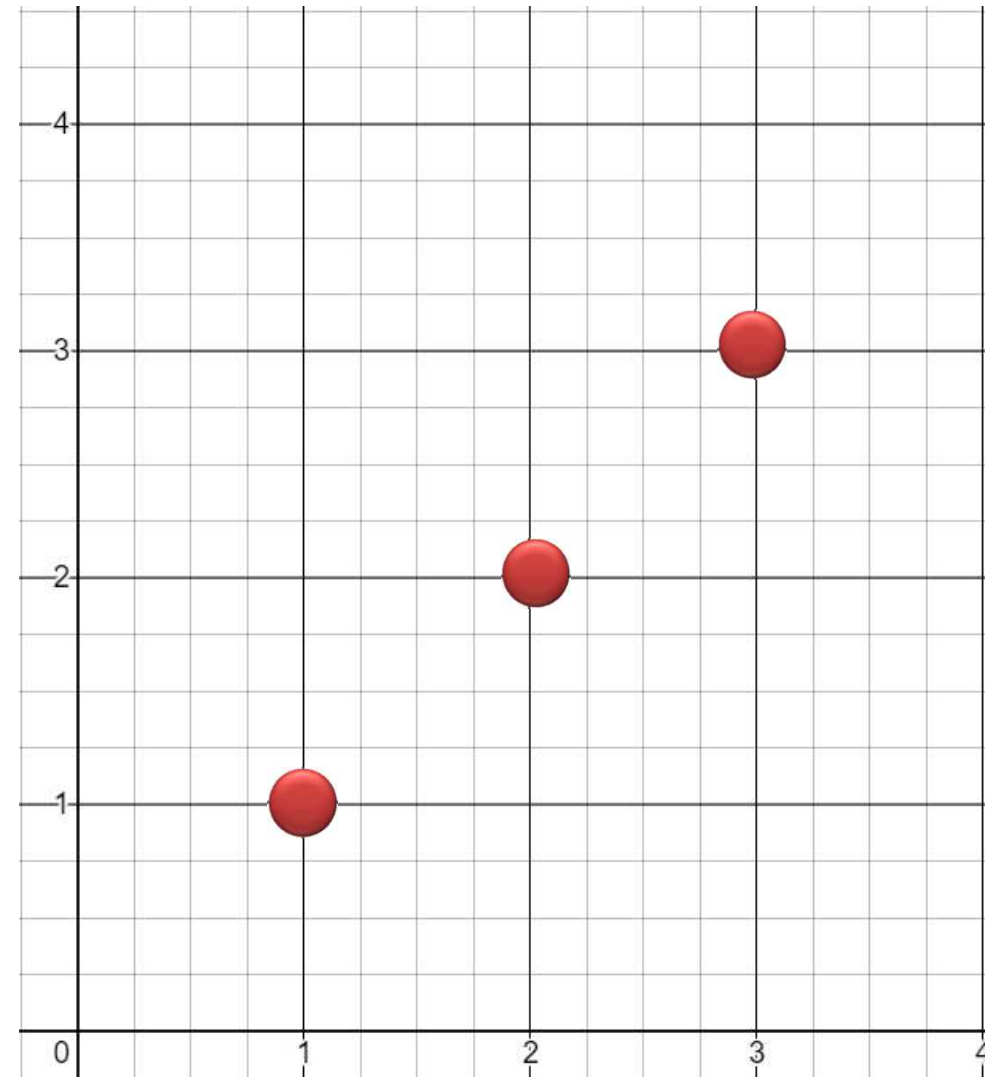
X (Feature)	Y(Label)
1	1
2	2
3	3



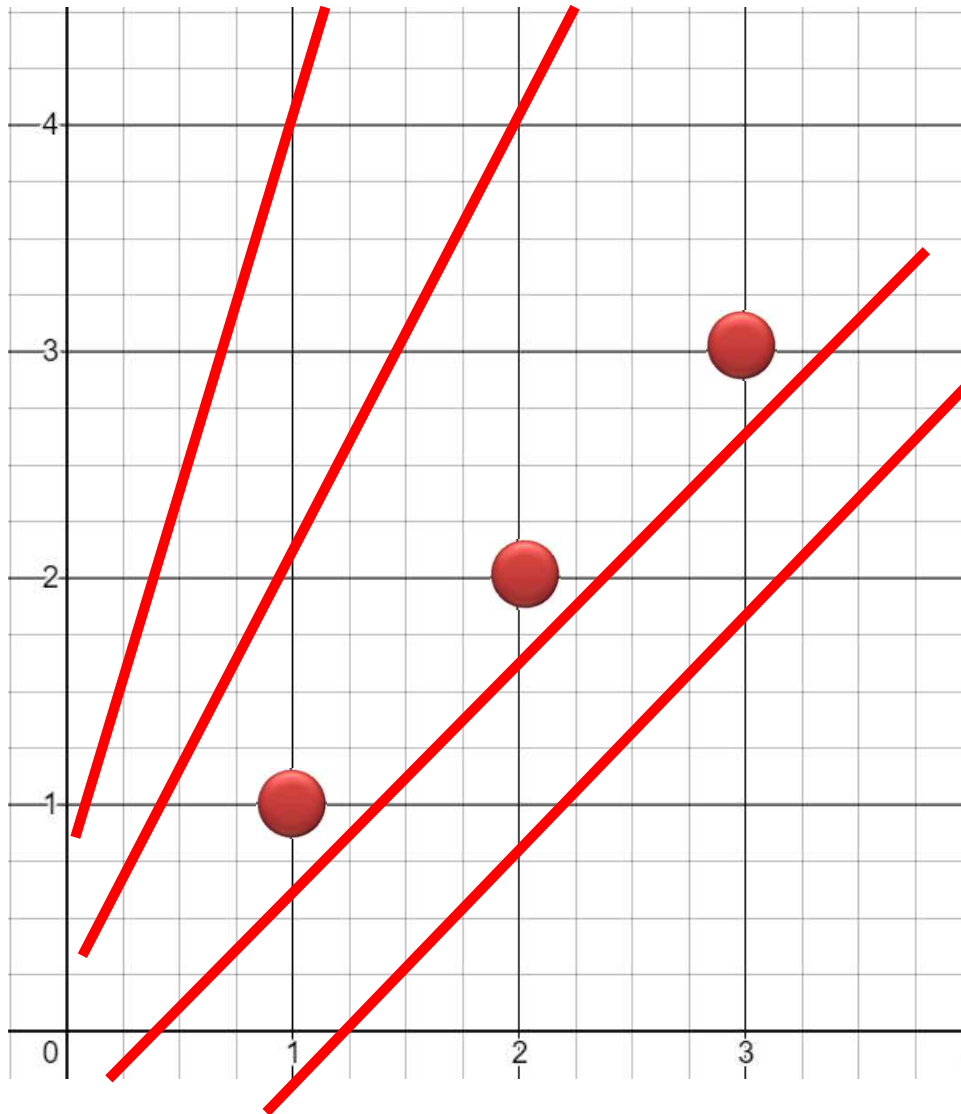
Regression Modeling 절차 (2)

X (Feature)	Y(Label)
1	1
2	2
3	3

Presentation



Hypothesis(가설) 수립



- ✓ **Linear regression**에서 예상되는 모델의 가설(H)을 수립하고, 수립된 가설을 기반을 학습데이터를 입력하여 선의 식을 찾아낸다.

$$H(X) = Wx + b$$



Hypothesis(가설)

Regression Modeling 절차 (4)

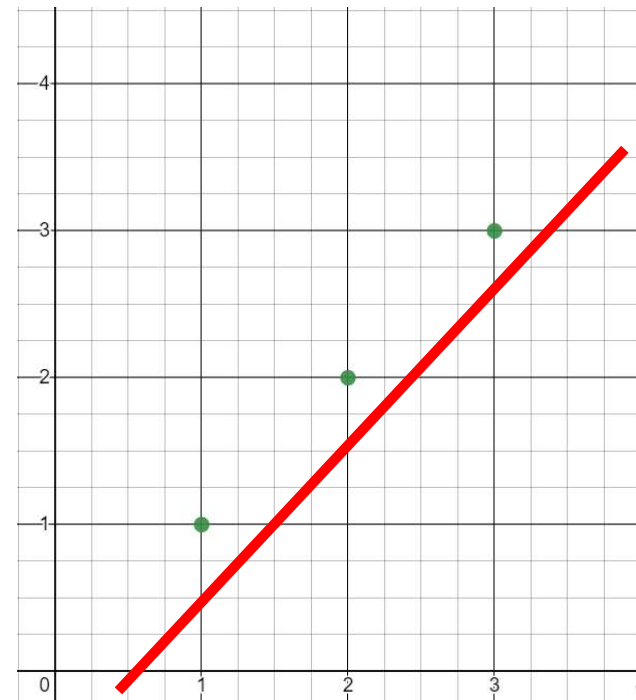
- 정답에 근접하는 가설을 찾는다.

$$H(X) = Wx + b$$

Training variable

Input data:
- 학습 시 학습 데이터 입력

X (Feature)	Y(Label)
1	1
2	2
3	3

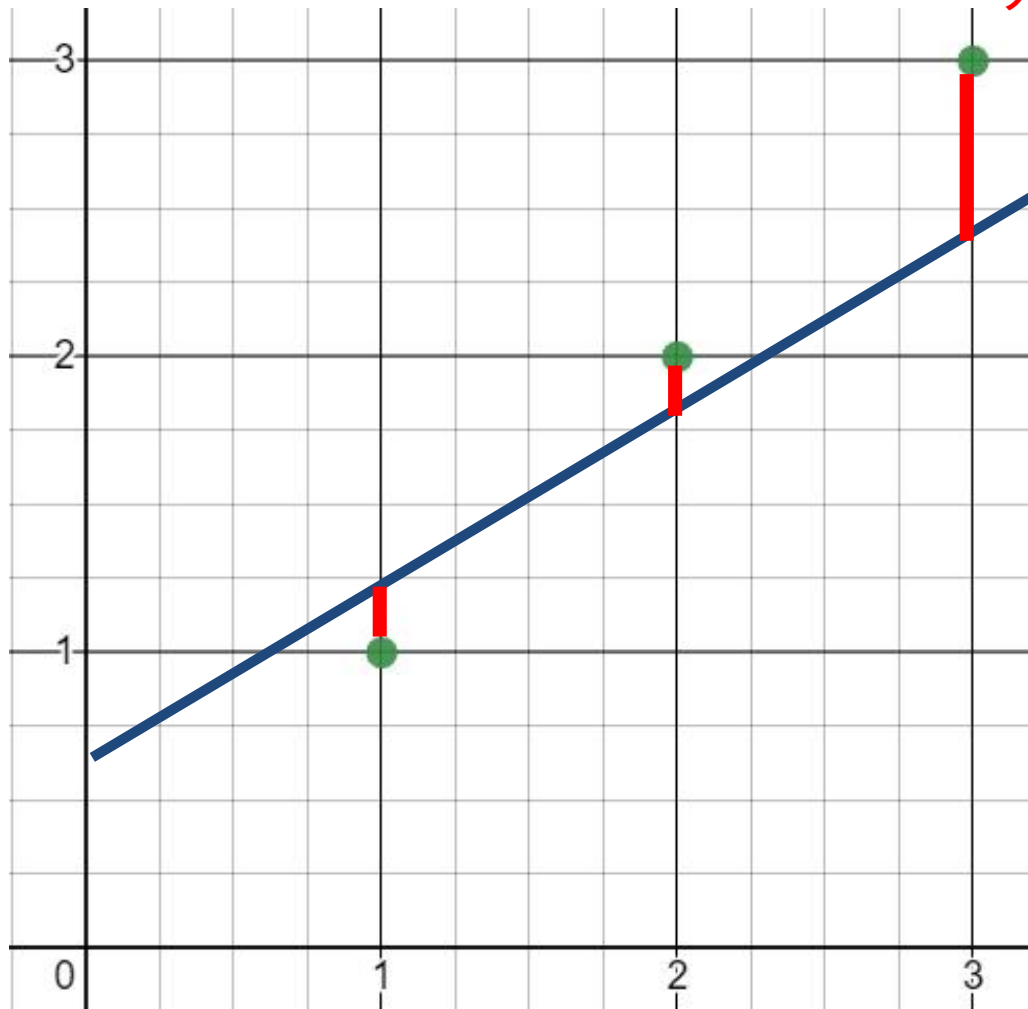


Regression Modeling 절차 (5)

- 정답에 근접하는 가설을 어떻게 찾는가?

$$H(X) = W x + b$$

✓ 학습데이터 입력 후 결과값과 정답과의
거리가 짧을 수록 정답에 가까움
거리가 멀면 멀수록 정답에서 멀어짐



Regression Modeling 절차 (6)

- 정답에 근접하는 가설을 어떻게 찾는가?

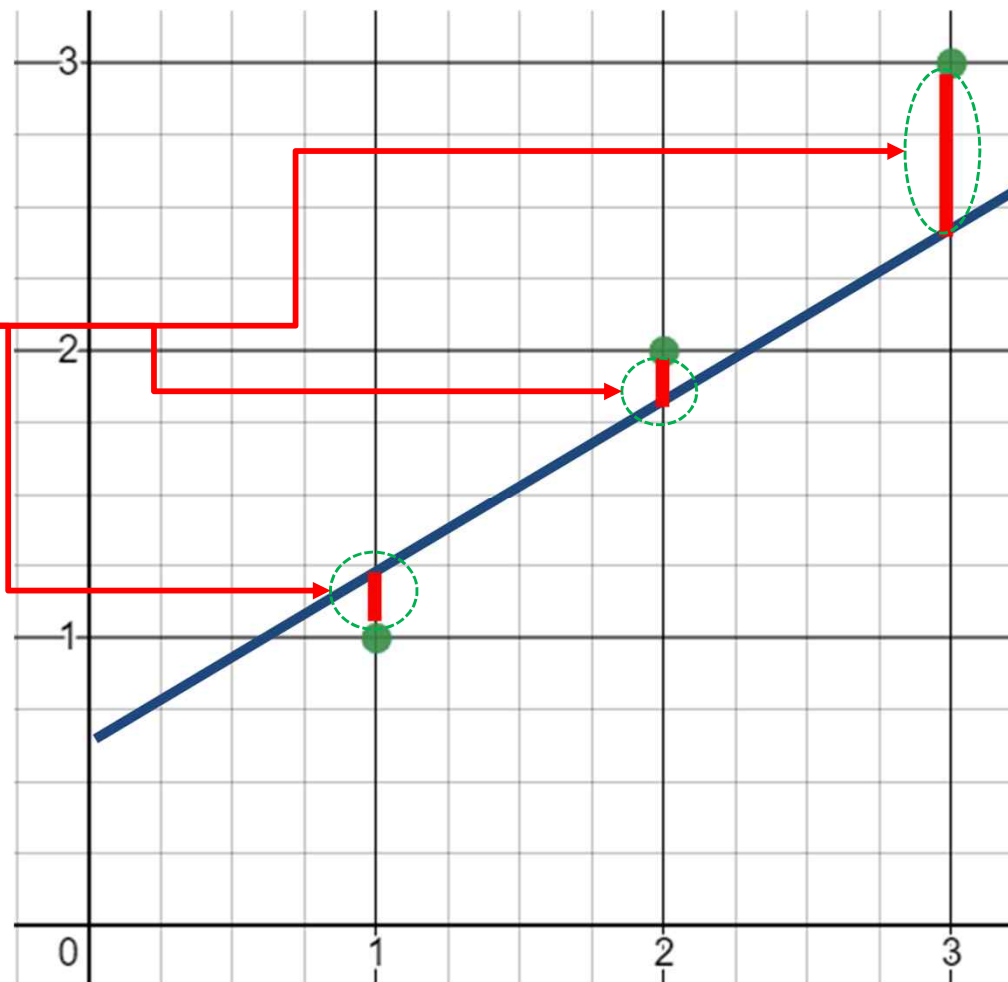
Cost function [Loss function] 을 활용하여 정답에 근접하는 가설 값(x, b)을 구함.

$$H(X) = W x + b$$

$$H(x) - y$$



$$(H(x) - y)^2$$



Regression Modeling 절차 (7)

- **Cost function**

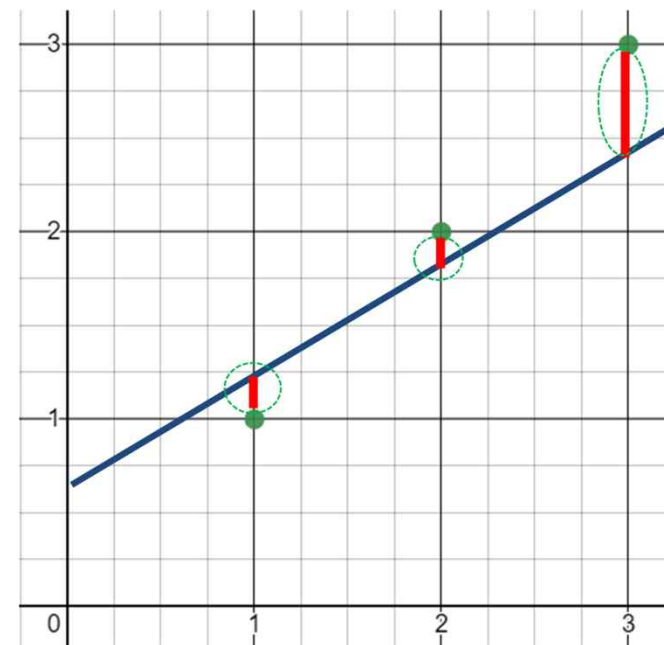
코스트 함수(Cost Function)는 머신러닝과 최적화 문제에서
모델의 예측값과 실제값 사이의 차이를 수치화하는 함수

$$\frac{(H(x^1) - y^1)^2 + (H(x^2) - y^2)^2 + (H(x^3) - y^3)^2}{3}$$



$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$

$$cost = \frac{\sum_{i=1}^m (H(x^i) - y^i)^2}{m}$$



Regression Modeling 절차 (8)

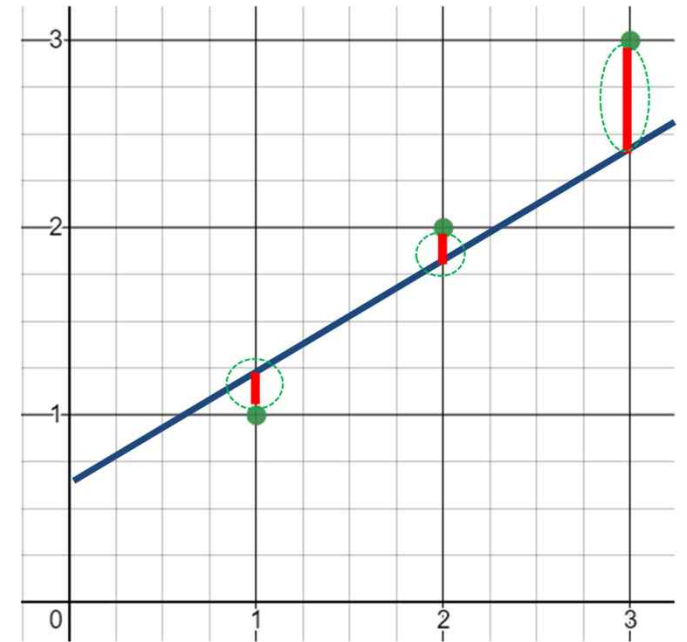
- Cost function

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$

$$H(X) = W x + b$$



$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$



Regression Modeling 절차 (9)

- Cost function

$$\text{Minimize } \textit{cost} (W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$

가장 작은 Cost 값을 가지는 W, b 를 찾는 것이 학습을 통해서 이루어짐!!

어떻게 Cost function의 최소 값을 찾을까?

Simplified hypothesis

$$H(X) = W x$$

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$

Calculate the value of cost(W)!! (1)

$$H(x) = W x$$

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$

X	Y(Label)
1	1
2	2
3	3

✓ When $W = 0$

$$\frac{(0 \times 1 - 1)^2 + (0 \times 2 - 2)^2 + (0 \times 3 - 3)^2}{3} = 4.67$$

✓ When $W = 1$

$$\frac{(1 \times 1 - 1)^2 + (1 \times 2 - 2)^2 + (1 \times 3 - 3)^2}{3} = 0$$


✓ When $W = 2$

$$\frac{(2 \times 1 - 1)^2 + (2 \times 2 - 2)^2 + (2 \times 3 - 3)^2}{3} = 4.67$$

Find out the value of minimized cost(W)!! (2)

```
✓ [11] import matplotlib.pyplot as plt
```

```
✓ ▶ x = [1, 2, 3]  
y = [1, 2, 3]  
  
x_axis = []  
y_axis = []  
for w in range(21):  
    w = w * 0.1  
    result = ((w * x[0] - y[0])**2 + (w * x[1] - y[1])**2 + (w * x[2] - y[2])**2)/3  
    x_axis.append(w)  
    y_axis.append(result)  
  
plt.plot(x_axis, y_axis)  
plt.show()
```

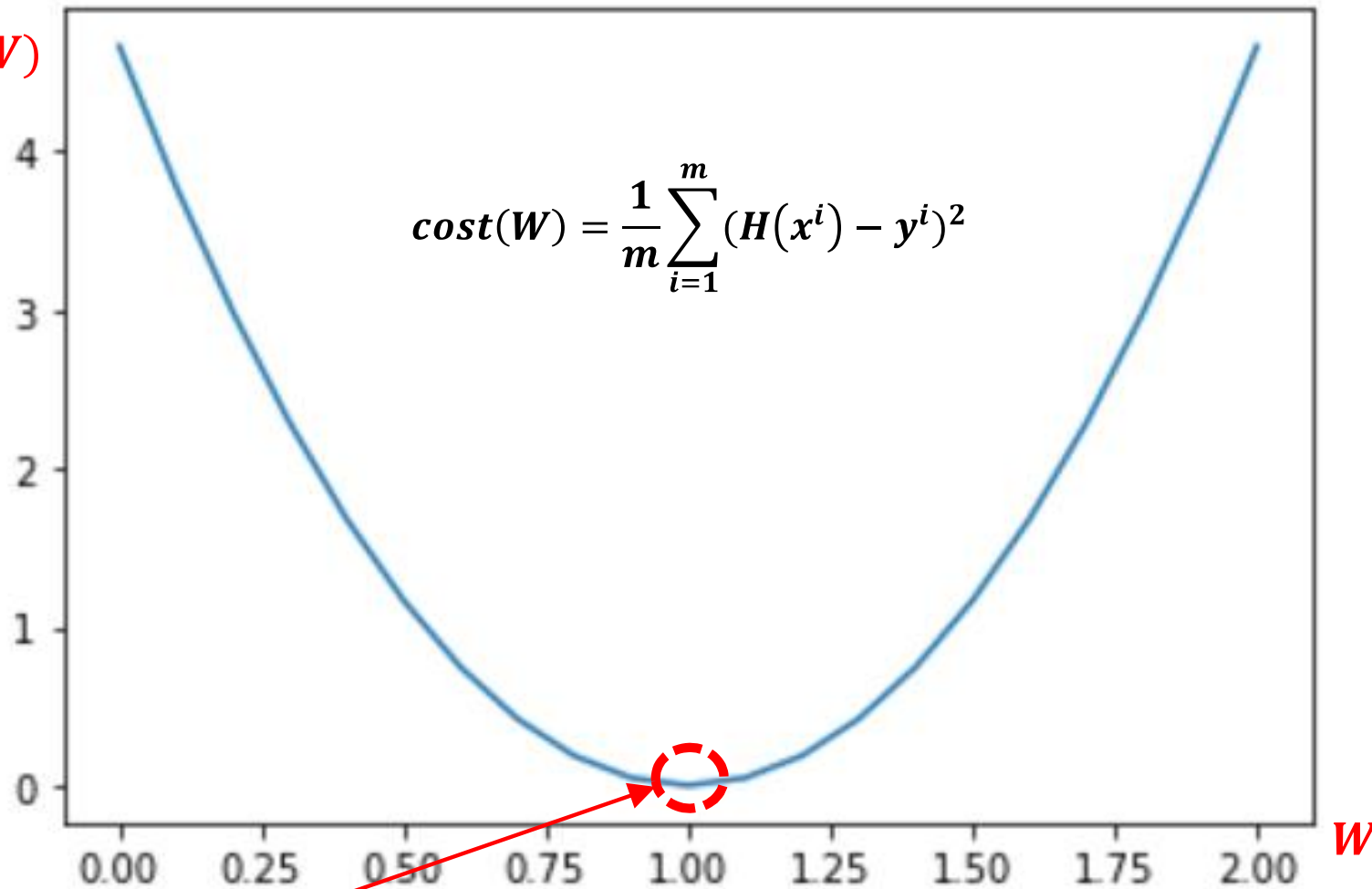

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2$$

Calculate the value of cost(W)! (2)

Remind!!

- Cost function(손실 함수)은 주어진 기울기와 절편이 예측한 값과 실제 정답 값(타겟 값) 사이의 오차를 측정하여 평균적으로 얼마나 차이가 나는지를 나타내는 함수
- 즉! Cost function의 값이 0에 가까울수록 예측 값이 정답과 가깝다는 의미이며, 이를 최소화해야 최적의 기울기와 절편을 찾을 수 있다.

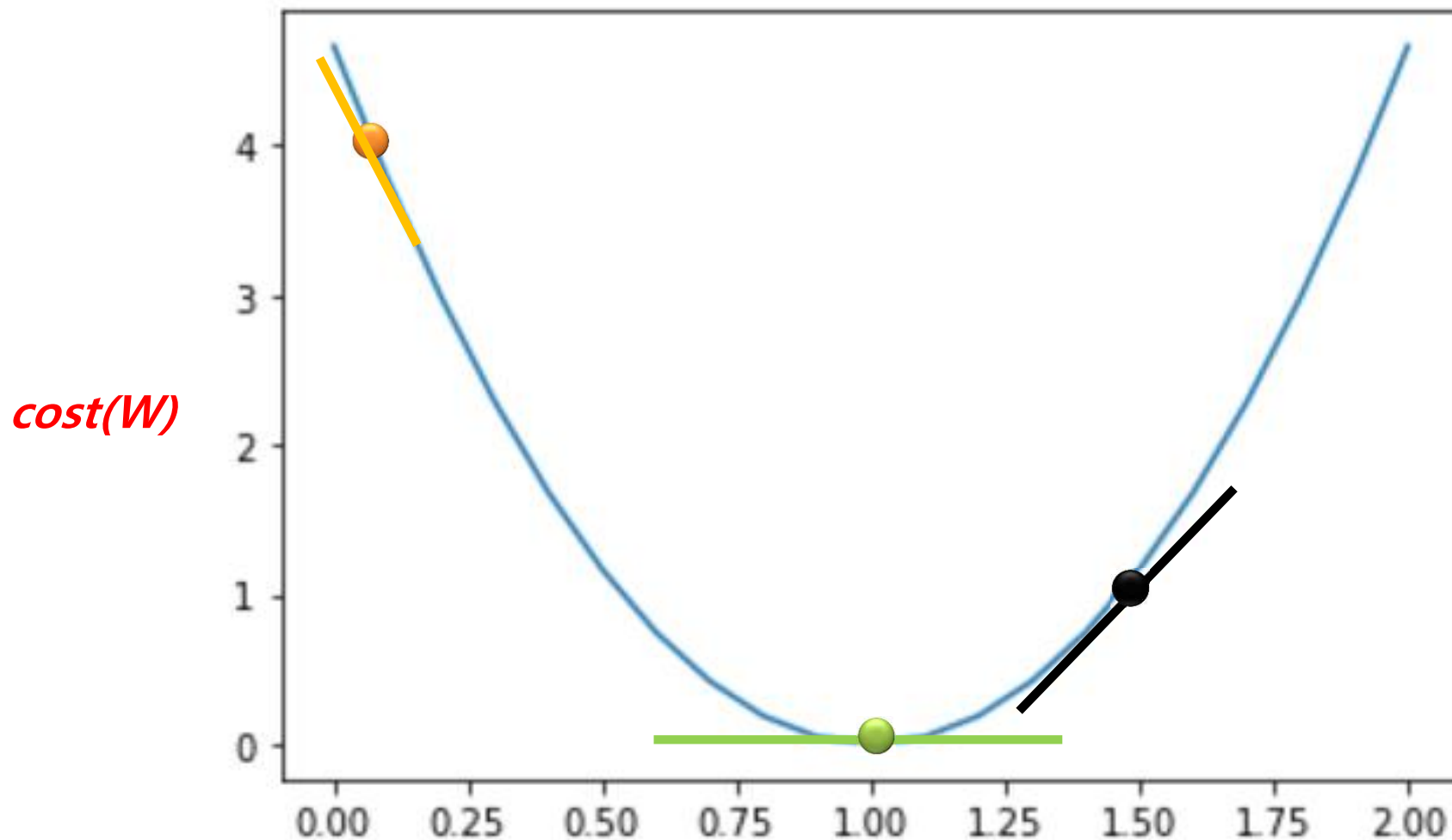
cost (W)



최저점을 어떻게 찾아 낼 수 있을까?

경사하강 (Gradient Descent) 알고리즘 (1)

- 경사하강법은 기울기(Gradient) 를 이용하여 함수의 최소값을 찾아가는 방식
- Key points**
 - 기울기(Gradient) 값을 이용하여 최적화 수행
 - 기울기 값이 0인 경우: 최적점(극값)에 도달한 상태
 - 기울기 값이 음수인 경우: 값이 증가하는 방향으로 이동
 - 기울기 값이 양수인 경우: 값이 감소하는 방향으로 이동



경사하강 (Gradient Descent) 알고리즘 (2)

- 경사하강법

$$W := W - \alpha \frac{\partial \text{cost}(W)}{\partial W}$$



$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^m 2(W(x^i) - y^i)x^i$$



$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (W(x^i) - y^i)x^i$$

$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (H(x^i) - y^i)^2$$



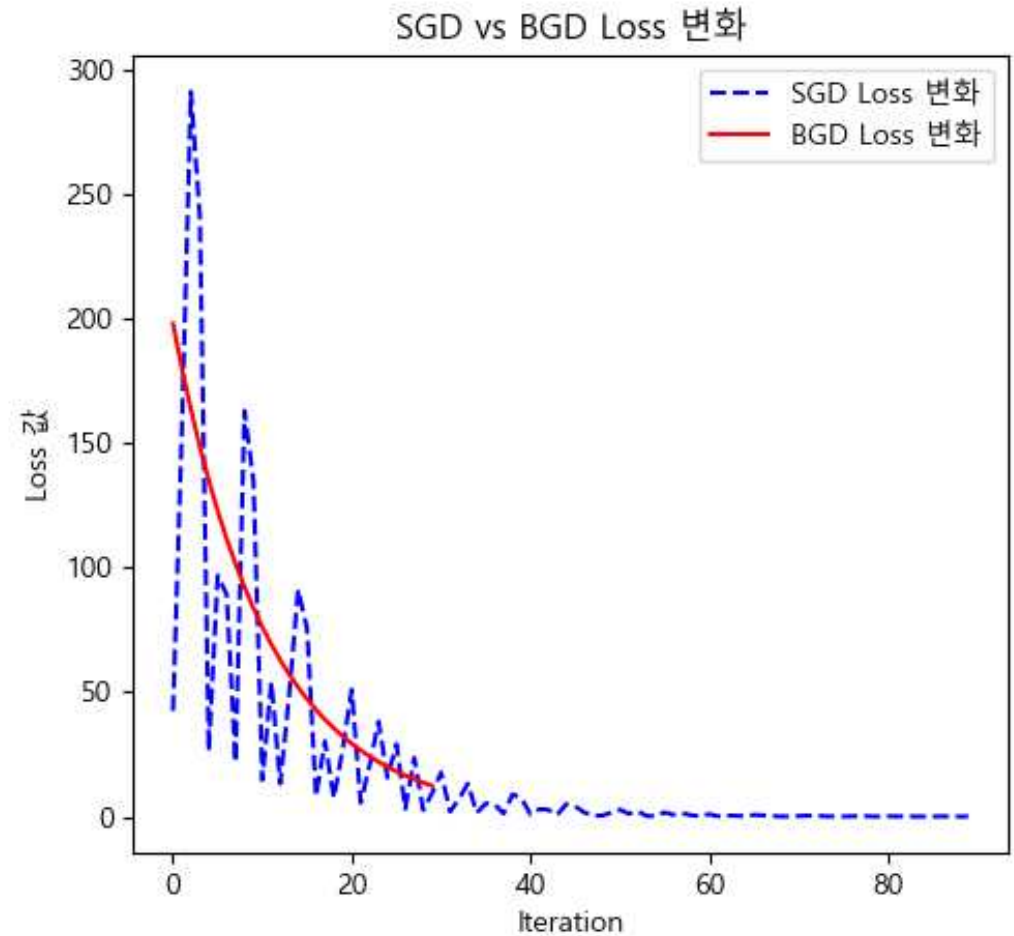
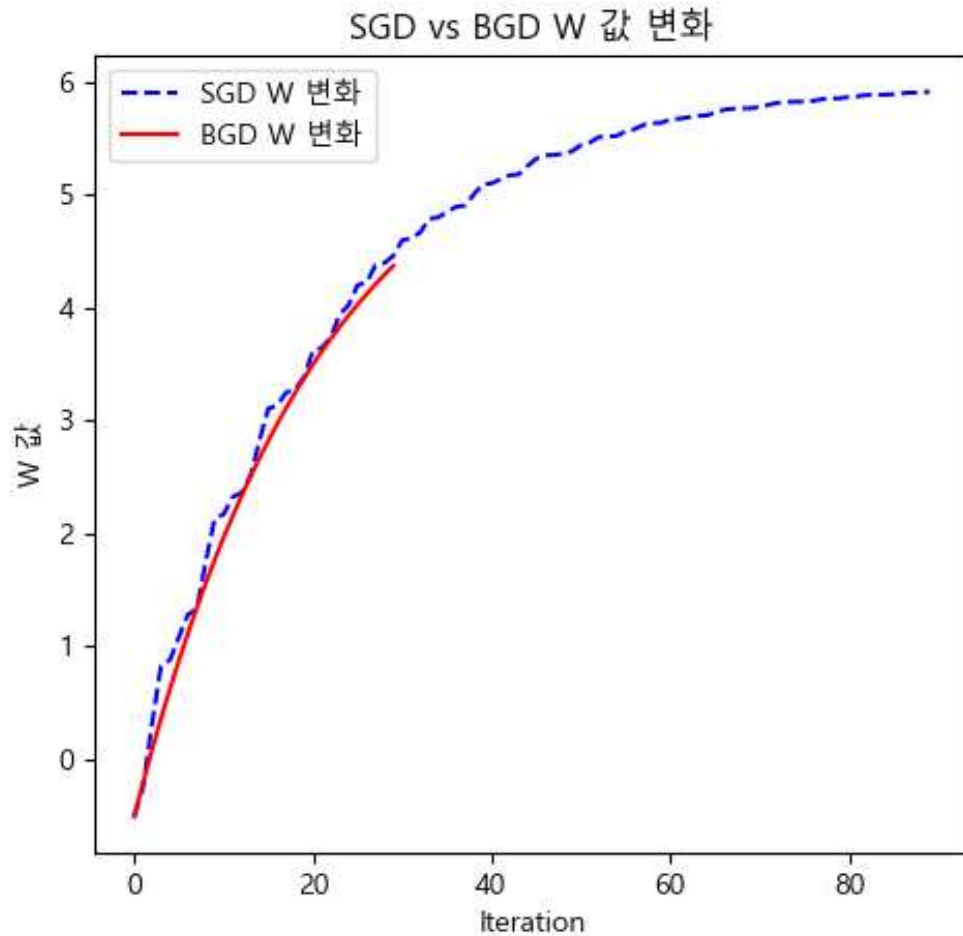
$$2x(xw - y)$$

Lab 1 Linear Regression 구현

Stochastic Gradient Descent(SGD) Vs. Batch Gradient Descent(BGD)

구분	확률적 경사 하강법 (SGD)	배치 경사 하강법 (BGD)
데이터 사용 방식	<ul style="list-style-type: none">• 한 개의 샘플을 사용하여 가중치 업데이트	<ul style="list-style-type: none">• 전체 데이터셋을 사용하여 가중치 업데이트
연산 속도	<ul style="list-style-type: none">• 빠름 (업데이트가 자주 발생)	<ul style="list-style-type: none">• 느림 (한 번의 업데이트에 많은 연산 필요)
수렴 속도	<ul style="list-style-type: none">• 초기 수렴이 빠름• 변동성이 큼	<ul style="list-style-type: none">• 수렴 속도가 상대적으로 느림• 변동성이 적음
안정성	<ul style="list-style-type: none">• 노이즈가 많아 최적 정답에 도달하기 어려울 수 있음	<ul style="list-style-type: none">• 안정적으로 최적 정답에 수렴 가능
메모리 사용량	<ul style="list-style-type: none">• 적음 (한 샘플만 사용)	<ul style="list-style-type: none">• 많음 (전체 데이터셋 사용)

SGD vs BGD : epoch에 따른 W 값과 Loss의 변화,



- 데이터가 크고 실시간 학습이 필요하다면 → SGD
- 안정적인 최적화와 정확도가 중요하다면 → BGD
- 절충안으로 미니배치 경사 하강법(Mini-batch GD)도 고려 가능

SGD (Stochastic Gradient Descent) Version

Linear Regression + Gradient Descent 구현 (1)

```
1 import random
2 import matplotlib.pyplot as plt
3
4 # 학습 데이터
5 x_train = [1, 2, 3]
6 y_train = [6, 12, 18]
7
8 # 학습률 및 반복 횟수 설정
9 learning_rate = 0.01 # 안정적인 학습을 위해 작은 값 사용
10 epochs = 100 # 전체 데이터를 100번 학습
11
12 # 초기 가중치 설정 (무작위 작은 값)
13 w = random.uniform(-1, 1)
14
15 # 기록 저장 리스트
16 weights_history = []
17 loss_history = []
```

SGD (Stochastic Gradient Descent) Version

Linear Regression + Gradient Descent 구현 (2)

```
19 # Stochastic Gradient Descent (SGD) 구현
20 for epoch in range(epochs):
21     data = list(zip(x_train, y_train))
22     random.shuffle(data) # 데이터 섞기
23
24     for x, y in data:
25         # 1. 개별 데이터 샘플에 대한 기울기 계산 (누적 x)
26         gradient = x * (w * x - y)
27
28         # 2. 가중치 업데이트
29         w -= learning_rate * gradient
30
31         # 3. 새로운 가중치로 예측 후 손실 계산 (MSE)
32         loss = (w * x - y) ** 2
33
34         # 4. 값 저장
35         weights_history.append(w)
36         loss_history.append(loss)
37
38     # 5. Epoch 단위 출력 (마지막 샘플 기준)
39     print(f"SGD Epoch {epoch + 1}: W = {w:.5f}, Loss = {loss:.5f}")
40
41 print(f"최종 W (SGD): {w:.5f}")
```

SGD (Stochastic Gradient Descent) Version

Linear Regression + Gradient Descent 구현 (3)

```
43 # 그래프 시각화
44 plt.rc('font', family="Malgun Gothic")
45 plt.rcParams['axes.unicode_minus'] = False
46 plt.figure(figsize=(12, 5))
47
48 plt.subplot(1, 2, 1)
49 plt.plot(weights_history, label="SGD W 변화", linestyle='dashed')
50 plt.xlabel("Iteration")
51 plt.ylabel("W 값")
52 plt.title("Gradient Descent 과정에서 W 값 변화")
53 plt.legend()
54
55 plt.subplot(1, 2, 2)
56 plt.plot(loss_history, label="SGD Loss 변화", linestyle='dashed', color='blue')
57 plt.xlabel("Iteration")
58 plt.ylabel("Loss 값")
59 plt.title("Gradient Descent 과정에서 Loss 변화")
60 plt.legend()
61
62 plt.show()
```


BGD (Batch Gradient Descent) Version

Linear Regression + Gradient Descent 구현 (1)

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # ✅ 1. 데이터 생성 (리스트 활용, numpy 함수 최소화)
5 x_data = [np.random.rand() * 10 for _ in range(50)] # 입력값 X (0~10 범위의 난수 50개)
6 y_data = [2 * x + np.random.rand() * 4 for x in x_data] # 실제값 Y ( $y = 2x + \text{노이즈}$ )
7
8 # ✅ 2. 학습 파라미터 초기화
9 weight = 4 # 초기 가중치 (W)
10 learning_rate = 0.01 # 학습률
11 epochs = 100 # 학습 반복 횟수
12 loss_history = [] # 손실 함수 값 저장 리스트
```

```
14 # ✅ 3. 배치 경사 하강법 (Batch Gradient Descent, BGD)
15 for _ in range(epochs):
16     total_loss = 0        # 손실 값 초기화
17     gradient_sum = 0      # 기울기(Gradient) 합 초기화
18
19     for i in range(len(x_data)): # 개별 데이터 처리
20         x_train = x_data[i]
21         y_train = y_data[i]
22
23         # (1) 예측값 계산
24         predicted_y = weight * x_train
25
26         # (2) 오차(error) 계산
27         error = predicted_y - y_train
28
29         # (3) 손실 값 계산 (MSE를 위한 합산)
30         total_loss += error ** 2
31
32         # (4) 평균 기울기(Gradient) 계산을 위한 합산
33         gradient_sum += x_train * error
34
35     # (5) 평균 손실 계산 (MSE)
36     total_loss /= len(x_data) # 전체 개수로 나눠 평균 손실 계산
37
38     # (6) 평균 기울기 계산
39     gradient = gradient_sum / len(x_data)
40
41     # (7) 가중치(weight) 업데이트 (경사 하강법 적용)
42     weight = weight - learning_rate * gradient
43
44     # (8) 손실 값 저장 (시각화를 위해)
45     loss_history.append(total_loss)
```

BGD (Batch Gradient Descent) Version

Linear Regression + Gradient Descent 구현 (2)

```
19 # Batch Gradient Descent (BGD) 구현
20 for epoch in range(epochs):
21     gradient = 0.0 # 기울기 초기화
22
23     # 1. 기울기(gradient) 계산
24     for x, y in zip(x_train, y_train):
25         gradient += x * (w * x - y)
26     gradient /= len(x_train) # 평균 기울기 계산
27
28     # 2. 가중치 업데이트
29     w -= learning_rate * gradient
30
31     # 3. 새로운 가중치로 예측 후 손실 계산 (MSE)
32     loss = sum((w * x - y) ** 2 for x, y in zip(x_train, y_train)) / len(x_train)
33
34     # 4. 값 저장 및 출력
35     weights_history.append(w)
36     loss_history.append(loss)
37
38     print(f"BGD Epoch {epoch + 1}: W = {w:.5f}, Loss = {loss:.5f}")
39
40 print(f"최종 W (BGD): {w:.5f}")
```


BGD (Batch Gradient Descent) Version

Linear Regression + Gradient Descent 구현 (3)

```
47 # ✅ 4. 최적화된 가중치 출력
48 print(f"최적화된 가중치 (Weight): {weight}")
49
50 # ✅ 5. 결과 시각화
51 plt.figure(figsize=(10, 4))
52 plt.rc("font", family="Malgun Gothic")
53 plt.rc("axes", unicode_minus=False)
54
55 # 1 원본 데이터 및 학습된 회귀선 그래프
56 plt.subplot(1, 2, 1) # 첫 번째 서브플롯
57 plt.scatter(x_data, y_data, label="실제 데이터") # 원본 데이터 산점도
58 plt.plot(x_data, [weight * x for x in x_data], color='red', label="최적화된 선") # 학습된 회귀선
59 plt.legend()
60 plt.title("선형 회귀 결과")
61 plt.xlabel("x_data")
62 plt.ylabel("y_data")
63 plt.grid()
64
65 # 2 학습 과정에서 손실 감소 그래프
66 plt.subplot(1, 2, 2) # 두 번째 서브플롯
67 plt.plot(loss_history, label="MSE 손실 값", color="blue")
68 plt.xlabel("Epochs")
69 plt.ylabel("Loss (MSE)")
70 plt.title("손실 감소 그래프")
71 plt.grid()
72 plt.legend()
73
74 plt.show()
```

Scikit-Learn : 사이킷런

- 사이킷런(Scikit-Learn)은 파이썬 기반의 머신러닝 라이브러리로, 다양한 머신러닝 알고리즘을 쉽게 사용할 수 있도록 지원
- 지도학습, 비지도학습, 데이터 전처리, 모델 평가 등의 기능 포함
- NumPy, SciPy, Matplotlib과 같은 과학 연산 라이브러리와 호환
- 주요 특징
 - 다양한 머신러닝 알고리즘 제공 (회귀, 분류, 군집화, 차원 축소 등)
 - 일관된 API 구조
 - 효율적인 데이터 처리 지원 (Sparse Matrix 및 NumPy 활용)
 - 강력한 전처리 기능
 - 쉬운 하이퍼파라미터 튜닝 및 모델 평가 기능
- 설치 방법

```
pip install scikit-learn
```

샘플 코드 (1)

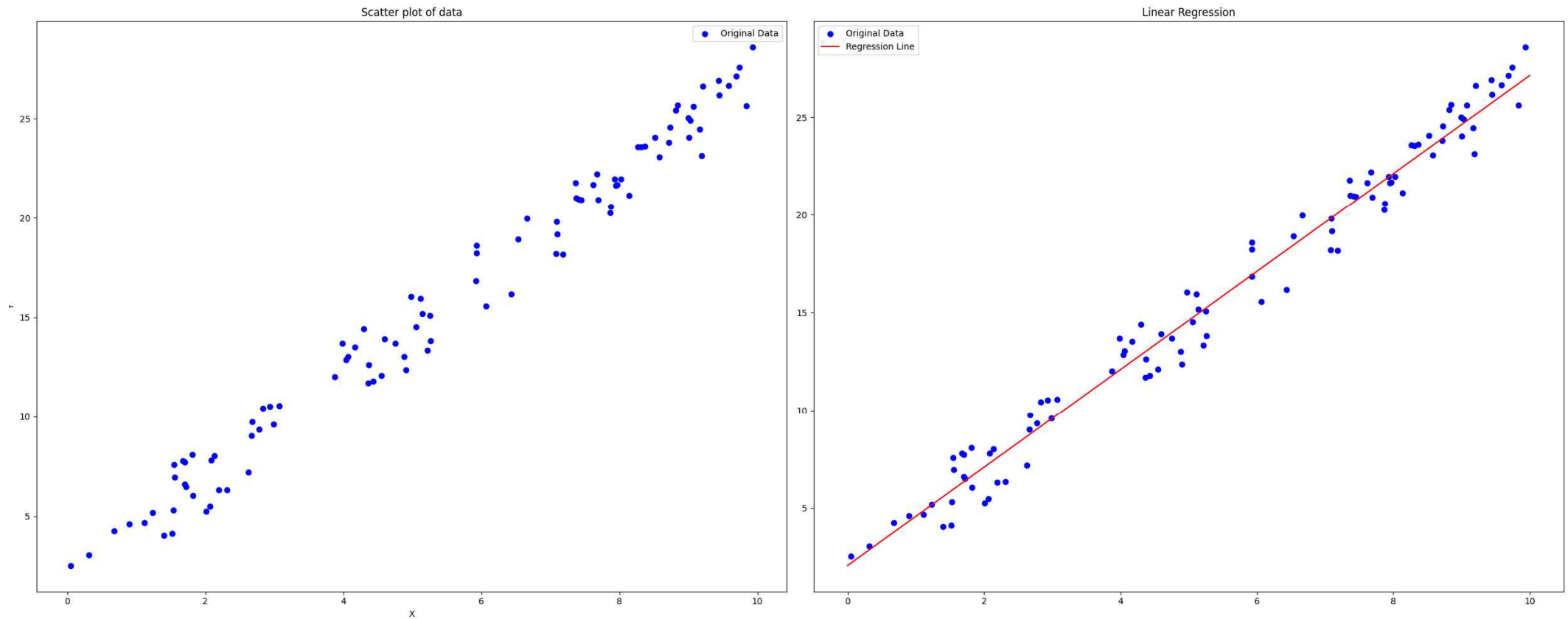
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import mean_squared_error
6
7 # 데이터 생성
8 x = np.random.rand(100, 1) * 10
9 y = 2.5 * x + np.random.rand(100, 1) * 4
10
11 # 데이터 분할
12 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
13
14 # 모델 학습
15 model = LinearRegression()
16 model.fit(x_train, y_train)
17
18 # 예측 및 평가
19 y_pred = model.predict(x_test)
```

샘플 코드 (2)

```
21 # 플롯 준비
22 fig, axes = plt.subplots(1, 2, figsize=(25, 10))
23
24 # 원본 데이터 산점도
25 axes[0].scatter(x, y, color='blue', label='Original Data')
26 axes[0].set_title('Scatter plot of data')
27 axes[0].set_xlabel('X')
28 axes[0].set_ylabel('Y')
29 axes[0].legend()
30
31 # 회귀선 및 데이터 산점도
32 x_val = np.linspace(0, 10, 100)
33 y_val = model.coef_[0, 0] * x_val + model.intercept_[0]
34
35 axes[1].scatter(x, y, color='blue', label='Original Data')
36 axes[1].plot(x_val, y_val, color='red', label='Regression Line')
37 axes[1].set_title('Linear Regression')
38 axes[1].legend()
```

샘플 코드 (3)

```
40 plt.tight_layout()
41 plt.show()
42
43 # 회귀식 출력
44 mse = mean_squared_error(y_test, y_pred)
45 print(f"MSE: {mean_squared_error(y_test, y_pred)}, 회귀 계수: {model.coef_[0,0]}, 절편: {model.intercept_[0]}")
```



MSE: 1.1749129631803754, 회귀 계수: 2.507100490027469, 절편: 2.0679133664748957

Dataset 이란?

- 데이터셋(Dataset)은 머신러닝 모델 학습을 위해 사용되는 데이터의 집합을 의미
- 일반적으로 입력(features)과 출력(labels 또는 targets)으로 구성되며, 모델을 학습시키고 평가하는 데 사용

- 데이터셋의 주요 구성 요소

1. **입력 데이터 (Input Data)**

- 입력 변수 (Input Variables): 원본 데이터에서 측정된 속성
- 특성 (Features): 입력 변수를 가공하여 모델 학습에 적합한 형태로 변환한 데이터
 - 예: 날짜 데이터를 "연, 월, 일"로 분리, 텍스트 데이터를 벡터로 변환

2. **출력 데이터 (Output Data)**

- 레이블 (Label, 정답 값): 지도 학습에서 학습할 정답 값

3. **샘플 (Sample, 데이터 포인트)**

- 데이터셋에서 하나의 입력과 출력이 짝을 이루는 개별 데이터
- 하나의 샘플이 여러 개의 입력 변수와 하나의 출력 값을 가질 수 있음

4. **추가 정보 (Optional)**

- 메타데이터 (Metadata): 데이터 출처, 수집 날짜, 데이터 정제 상태 등의 정보
- 가중치 (Sample Weights): 특정 데이터 샘플에 대한 중요도를 부여할 때 사용
- ID 필드 (ID Fields): 개별 샘플을 식별하는 고유 키

➡ **지도학습(Supervised Learning) 데이터셋**

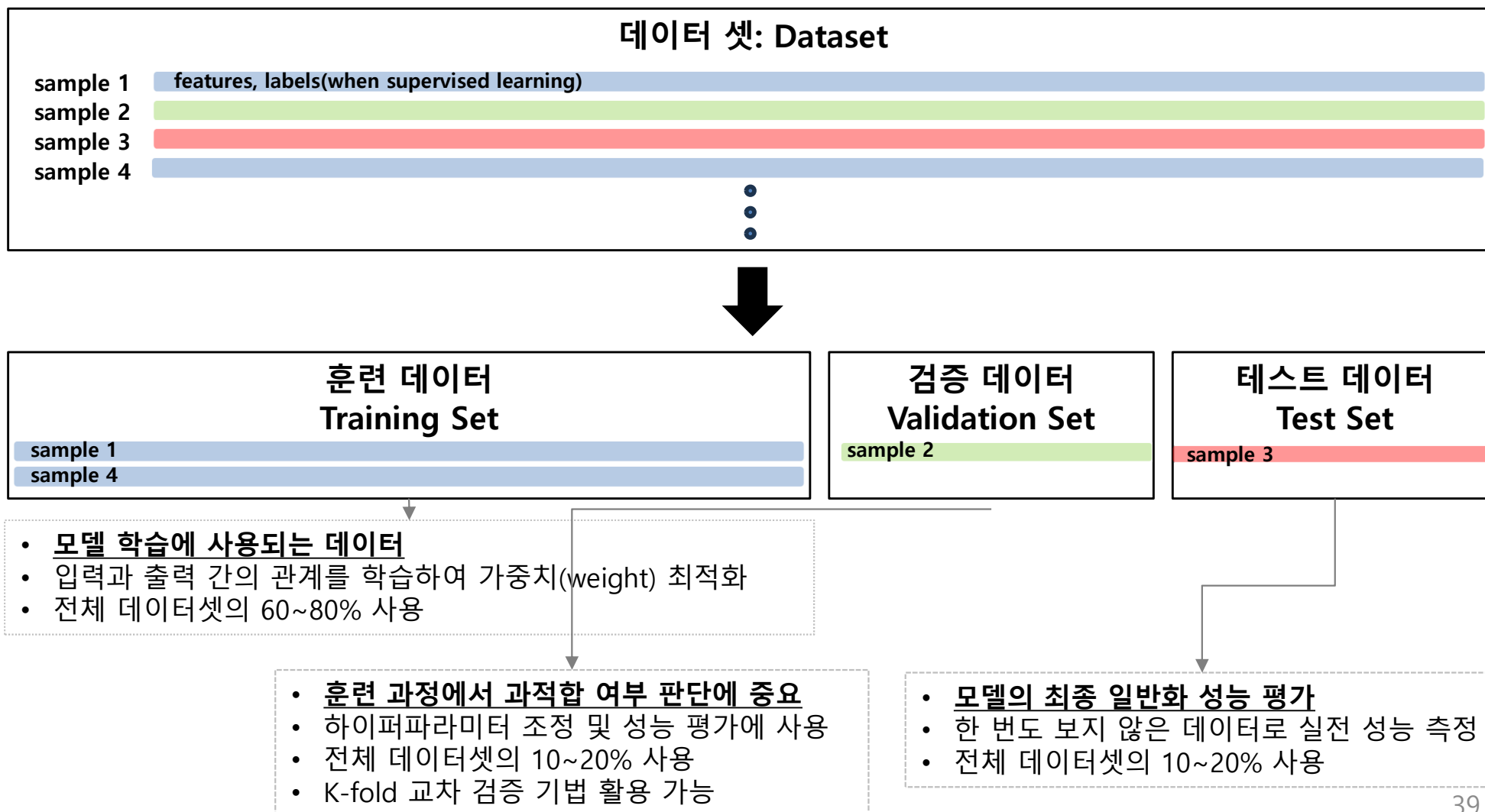
- 입력 데이터(X)와 정답(Y)이 포함됨
- 예: 이미지 분류 (이미지, 해당 클래스 라벨)

➡ **비지도학습(Unsupervised Learning) 데이터셋**

- 정답 없이 입력 데이터(X)만 존재
- 예: 군집화(Clustering)

Dataset 구성 및 분할

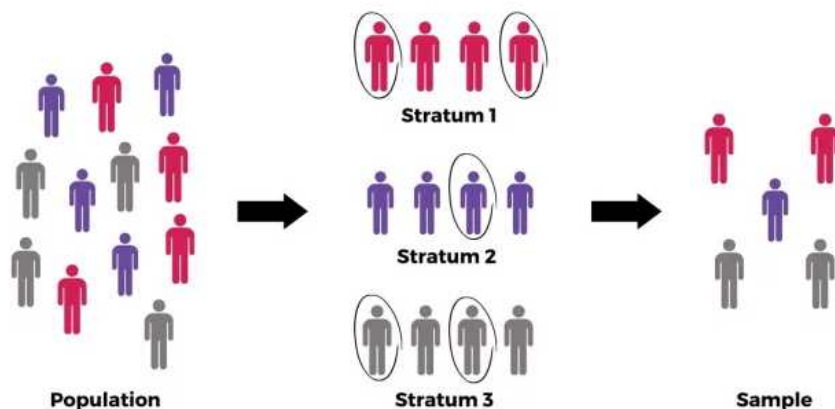
- 모델 성능을 올바르게 평가하기 위해 훈련(Train), 검증(Validation), 테스트(Test) 세 부분으로 분할
 - 훈련 데이터: 모델이 패턴을 학습하는 데 사용
 - 검증 데이터: 학습된 모델의 일반화 성능을 평가하고 과적합(Overfitting) 여부를 판단하여 최적의 하이퍼파라미터 선택에 활용
 - 테스트 데이터: 최종적으로 모델의 일반화 성능을 확인



Dataset 분할 방법 (1)

- 랜덤 샘플링 (Random Sampling)
 - 데이터를 무작위로 나누어 훈련, 검증, 테스트 데이터셋을 생성
 - 데이터가 충분히 많을 경우 일반적으로 사용됨
- 층화 샘플링 (Stratified Sampling)
 - 데이터의 클래스 분포를 유지하면서 훈련, 검증, 테스트 데이터셋을 나눔
 - 불균형한 데이터셋에서 필수적인 방법 (예: 의료 데이터, 금융 사기 탐지)
- K-Fold 교차 검증 (K-Fold Cross Validation)
 - 데이터를 K개의 부분(Fold)으로 나눈 후, K번 반복하여 학습과 평가 수행
 - 검증 데이터셋을 여러 번 변경하면서 모델의 일반화 성능을 향상시킴

• Stratified Sampling

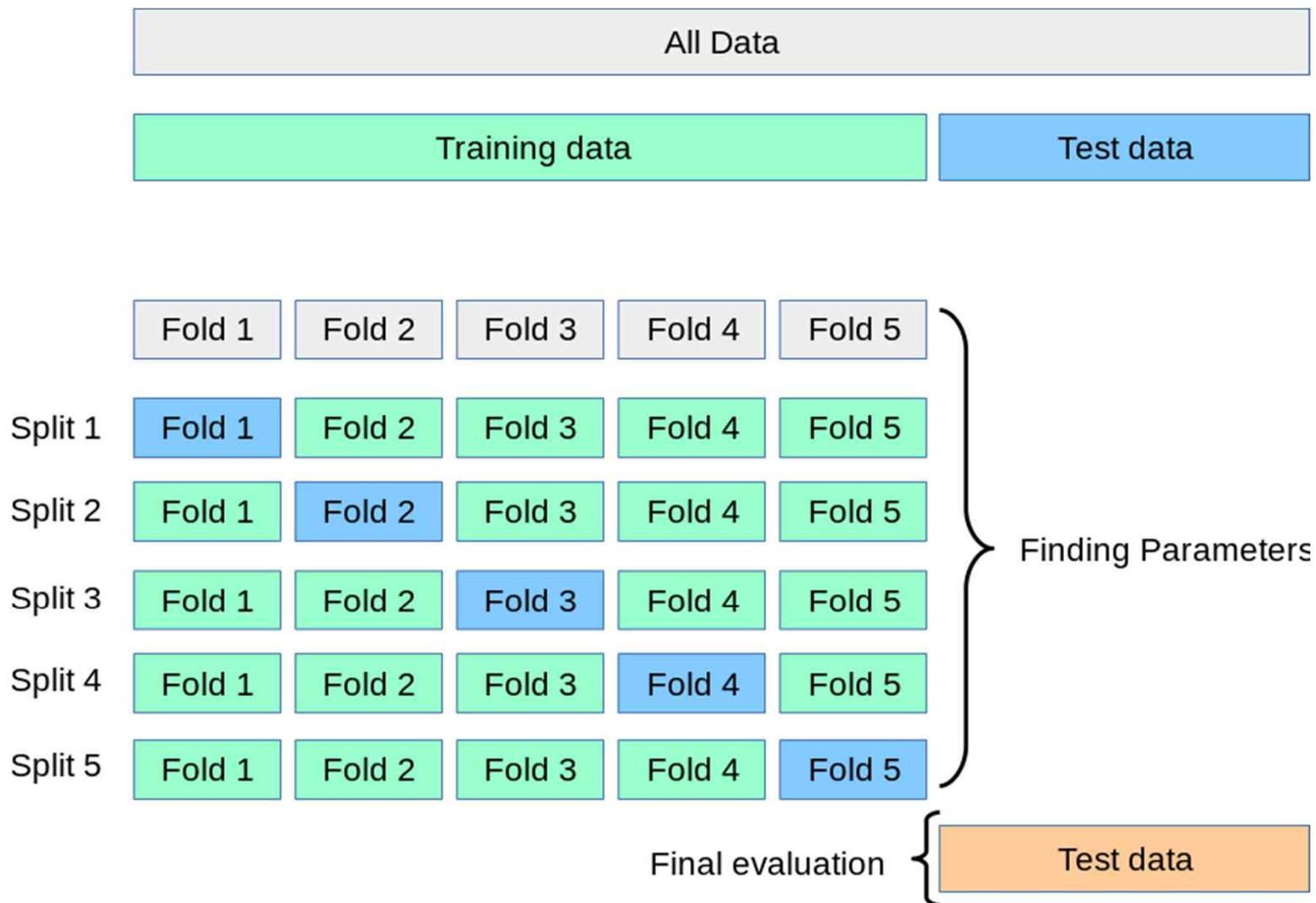


• Random Sampling



Dataset 분할 방법 (2)

- K Fold Cross Validation



사이킷런을 활용한 데이터셋 구성 방법

```
1 from sklearn.model_selection import train_test_split
2 import numpy as np
3
4 # 더미 데이터 생성
5 X = np.random.rand(1000, 10) # 1000개의 샘플, 10개의 특성
6 y = np.random.randint(0, 2, size=1000) # 0 또는 1의 이진 분류 라벨
7
8 # 80% 훈련 데이터, 20% 테스트 데이터 분할
9 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
10
11 print("훈련 데이터 크기:", X_train.shape)
12 print("테스트 데이터 크기:", X_test.shape)
```

```
1 from sklearn.model_selection import train_test_split
2 import numpy as np
3
4 # 더미 데이터 생성
5 X = np.random.rand(1000, 10) # 1000개의 샘플, 각 샘플당 10개의 특성
6 y = np.random.randint(0, 2, size=1000) # 0 또는 1의 이진 분류 라벨
7
8 # 70% 훈련, 15% 검증, 15% 테스트 데이터로 한 번에 분할
9 X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42) # 70% 훈련, 30% 나머지
10 X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42) # 15% 검증, 15% 테스트
11
12 # 데이터셋 크기 출력
13 print("훈련 데이터 크기:", X_train.shape) # (700, 10) → 70%
14 print("검증 데이터 크기:", X_val.shape) # (150, 10) → 15%
15 print("테스트 데이터 크기:", X_test.shape) # (150, 10) → 15%
```

Extra Slides

성능 지표 (Performance Metrics)

	실제 양성(Positive)	실제 음성(Negative)
예측 양성(Positive)	TP (True Positive)	FP (False Positive)
예측 음성(Negative)	FN (False Negative)	TN (True Negative)

- **정확도 (Accuracy)**

- 전체 샘플 중 올바르게 예측한 비율
- 계산식: $(TP + TN) / (TP + FP + TN + FN)$
- 클래스 불균형이 큰 경우 적절하지 않을 수 있음

- **정밀도 (Precision)**

- 모델이 양성(Positive)이라고 예측한 샘플 중 실제 양성 비율
- 계산식: $TP / (TP + FP)$
- FP(False Positive)를 줄이는 것이 중요한 경우 사용 (예: 스팸 필터)

- **재현율 (Recall, Sensitivity)**

- 실제 양성 샘플 중에서 모델이 올바르게 예측한 비율
- 계산식: $TP / (TP + FN)$
- FN(False Negative)을 줄이는 것이 중요한 경우 사용 (예: 암 진단)

- **F1-score**

- 정밀도와 재현율의 조화 평균
- 계산식: $(2 \times \text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$
- 불균형 데이터셋에서 성능 평가 시 유용

Q/A

감사합니다



주문식교육의 산실
영진전문대학교