머신러닝 심화 스터디 2차시

이론적 기초: 왜 머신러닝이 작동하는가?

Chapter 1: 왜 머신러닝이 작동하는가? 🤔

들어가며

1차 스터디에서 우리는 머신러닝이 **무엇인지**와 **어떻게 하는지**를 배웠습니다. 2차 스터디에서는 한 단계 더 깊이 들어가 **왜 작동하는지**를 이해해보겠습니다.

머신러닝을 배우다 보면 누구나 이런 의문을 가집니다:

> 🤔 "과거 데이터를 보고 학습했는데, 왜 미래를 예측할 수 있지?"
>
> 🤔 "학습한 데이터만 봤는데, 처음 보는 데이터도 맞출 수 있다고?"
>
> 🤔 "이게 정말 논리적으로 타당한 건가?"

이 질문들에 답하기 위해서는 머신러닝의 철학적, 수학적 기초를 이해해야 합니다.

1.1 귀납적 편향 (Inductive Bias) 🎯

귀납법이란?

먼저 논리학의 두 가지 추론 방법을 이해해야 합니다.

🔵 연역적 추론 (Deductive Reasoning):

```
전제1: 모든 사람은 죽는다
전제2: 소크라테스는 사람이다
결론: 소크라테스는 죽는다 ✅ (논리적으로 100% 확실)
```

🟡 귀납적 추론 (Inductive Reasoning):

```
관찰1: 태양이 어제도 동쪽에서 떴다
관찰2: 그제도 동쪽에서 떴다
관찰3: 지난 1만년 동안 계속 동쪽에서 떴다
결론: 내일도 태양은 동쪽에서 뜰 것이다 🤔 (확실하지 않음!)
```

> 💡 **핵심 차이**
>
> - **연역**: 전제가 참이면 결론도 100% 참
> - **귀납**: 관찰이 참이어도 결론은 "아마도" 참

머신러닝은 귀납적 추론이다

머신러닝이 하는 일을 다시 생각해봅시다:

> 관찰: 학습 데이터 $D = (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
> 목표: 새로운 입력 x_{new}에 대해 y_{new}를 예측하기

수식으로 표현:

> 💡 **수식 기호 설명 (처음 보는 분들을 위해)**
>
> - D = 데이터셋 (Dataset)
> - $\{ \}$ (중괄호) = **집합** (여러 개를 묶어놓은 것)
> - (x_1, y_1) = 하나의 **데이터 쌍** (입력-정답)
>   - x_1 (엑스 원): 첫 번째 입력 값
>   - y_1 (와이 원): 첫 번째 정답
> - ... = "그 사이에 더 많은 데이터가 있음"
> - n = 데이터 **개수** (예: $n = 1000$개)
> - 아래 첨자 (subscript): 순서를 나타냄 (1번째, 2번째, ...)

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

쉽게 풀어 쓰면:
"D라는 데이터셋에는 (입력, 정답) 쌍이 n개 들어있다"

목표: $x_{new} \rightarrow y_{new}$를 예측

문제:

- 우리는 x_{new}를 학습 중에 본 적이 **없습니다**
- 그런데 어떻게 y_{new}를 예측할 수 있을까요?

답:

- 어떤 **가정**을 하기 때문입니다!
- 이 가정을 **귀납적 편향(Inductive Bias)**이라고 부릅니다

귀납적 편향의 정의

> 📘 **귀납적 편향 (Inductive Bias)**
>
> 학습 알고리즘이 **본 적 없는 데이터**에 대해 예측할 때 사용하는 **사전 가정**들의 집합

쉽게 말하면:
"세상이 이럴 거야"라고 미리 가정하는 것입니다.

생활 속 비유 🏠

상황: 친구 집에 초대받았는데 현관문이 닫혀있습니다.

귀납적 편향 없이:

```
관찰: 이 집 현관문이 닫혀있다
결론: ??? (어떻게 해야 할지 전혀 모름)
        - 문을 두드려야 하나?
        - 창문으로 들어가야 하나?
        - 벽을 뚫어야 하나?
```

귀납적 편향 있으면:

```
가정: "보통 집은 문을 두드리면 열어준다"
행동: 문을 두드린다 ✅
```

머신러닝도 마찬가지입니다! **세상은 보통 이렇게 작동한다**는 가정이 있어야 본 적 없는 상황에 대응할 수 있습니다.

머신러닝 알고리즘의 귀납적 편향 예시

1️⃣ k-최근접 이웃 (k-NN)

귀납적 편향:

> "비슷한 입력은 비슷한 출력을 가진다"

수식적 표현:

$$\text{if } ||x_i - x_j|| \text{ is small, then } |y_i - y_j| \text{ is small}$$

예시:

```
학습 데이터:
- 키 170cm, 몸무게 65kg → 성인 남성
- 키 175cm, 몸무게 70kg → 성인 남성
- 키 120cm, 몸무게 30kg → 어린이

새 데이터:
- 키 173cm, 몸무게 68kg → ?

k-NN의 판단: "비슷한 키/몸무게를 가진 사람들은 성인 남성이었으니,
              이 사람도 성인 남성일 것이다" ✅
```

2 선형 회귀 (Linear Regression)

귀납적 편향:

> "입력과 출력 사이의 관계는 직선(선형)이다"

수식적 표현:

$$y = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$$

예시:

```
학습 데이터: 집 크기와 가격
- 30평 → 3억
- 40평 → 4억
- 50평 → 5억

새 데이터: 35평 → ?

선형 회귀의 판단: "직선 관계니까 3.5억일 것이다" ✅
```

> ⚠️ **만약 실제로는 2차 곡선 관계라면?**
>
> 선형 회귀는 **자기 가정(직선)** 때문에 틀릴 수 있습니다!

3️⃣ 결정 트리 (Decision Tree)

귀납적 편향:

> "데이터는 if-then 규칙으로 나눌 수 있다"

예시:

```
if 날씨 == 맑음:
    if 온도 > 25도:
        예측 = "아이스크림 많이 팔림"
    else:
        예측 = "아이스크림 조금 팔림"
else:
    예측 = "아이스크림 안 팔림"
```

"모든 백조는 하얗다" 문제 🦢

귀납적 편향의 위험성을 보여주는 유명한 철학적 예시입니다.

유럽인들의 관찰:

```
백조 #1: 하얗다
백조 #2: 하얗다
백조 #3: 하얗다
...
백조 #10000: 하얗다

귀납적 결론: "모든 백조는 하얗다" ✅
```

그런데...

호주에서 **검은 백조**가 발견되었습니다! 🦢⬛

교훈:

1. **귀납적 추론은 100% 확실하지 않다**
2. **관찰한 데이터 범위 밖에서는 틀릴 수 있다**
3. **"학습 데이터 분포"와 "실전 데이터 분포"가 다르면 실패한다**

머신러닝에서의 "검은 백조"

예시 1: 채용 AI의 편향

```
학습 데이터: 과거 10년간 합격한 지원자 (대부분 남성)
귀납적 편향: "좋은 지원자는 남성의 특징을 가진다"
결과: 여성 지원자를 불합격시킴 ❌
```

예시 2: 자율주행차

```
학습 데이터: 맑은 날씨의 도로 영상
귀납적 편향: "도로는 명확하게 보인다"
결과: 눈 오는 날 차선을 못 찾음 ❌
```

예시 3: 얼굴 인식 AI

```
학습 데이터: 주로 특정 인종의 얼굴
귀납적 편향: "얼굴은 이런 특징을 가진다"
결과: 다른 인종 얼굴 인식률 낮음 ❌
```

귀납적 편향의 강도 스펙트럼

머신러닝 알고리즘들을 귀납적 편향의 강도로 정렬하면:

```
강한 편향 ←──────────────────────→ 약한 편향
    ↓                                ↓
  선형회귀                          신경망
  결정트리 (얕은)                    k-NN
  나이브 베이즈                      결정트리 (깊은)
```

강한 편향 (Strong Bias):

- ✅ 장점: 적은 데이터로도 학습 가능, 빠름
- ❌ 단점: 가정이 틀리면 성능 나쁨, 유연성 낮음

약한 편향 (Weak Bias):

- ✅ 장점: 유연함, 복잡한 패턴 학습 가능
- ❌ 단점: 많은 데이터 필요, 느림

수학적 표현: 버전 공간 (Version Space)

왜 버전 공간 개념이 필요한가?

데이터를 맞추는 함수가 무한히 많기 때문입니다!

구체적 예시:
점 3개를 지나는 함수 찾기

```
데이터: (1, 2), (2, 4), (3, 6)

가능한 함수들:
- 함수 1: y = 2x (직선)
- 함수 2: y = 2x + 0.0001(x-1)(x-2)(x-3) (거의 직선)
- 함수 3: y = 2x + sin(100x)/1000 (미세하게 진동)
- 함수 4: ...
→ 무한히 많음!
```

이 중 어떤 함수를 선택해야 할까요?
→ **버전 공간**: "데이터를 맞추는 모든 후보 함수들"
→ **귀납적 편향**: "이 중 하나를 고르는 기준"

수식으로 정의:

> 💡 **수식 기호 설명 (집합론)**
>
> - $\in$ (원소 기호): "~에 속한다", "~의 멤버이다"
>   - 예: $h \in \mathcal{F}$ = "h는 F의 원소이다"
> - $|$ (수직선): "~인 것들 중에서" (조건)
> - $\forall$ (for all): "모든" ($\forall$는 거꾸로 된 A = All)
>   - 예: $\forall x$ = "모든 x에 대해"

$$\mathcal{H} = \{h \in \mathcal{F} \mid h(x_i) = y_i, \forall (x_i, y_i) \in D\}$$

쉽게 풀어 읽으면:
"$\mathcal{H}$는, $\mathcal{F}$에 속한 함수 h 중에서, 모든 학습 데이터 (x_i, y_i)에 대해 $h(x_i) = y_i$를 만족하는 함수들의 집합"

여기서:

- $\mathcal{F}$ (Calligraphic F): 알고리즘이 고려할 수 있는 **모든 함수** (가설 공간)
- D: 학습 데이터
- $\mathcal{H}$ (Calligraphic H): 학습 데이터와 맞는 가설들 (**버전 공간**)

문제:
$\mathcal{H}$에는 보통 **무한히 많은 가설**이 있습니다!

귀납적 편향의 역할:
$\mathcal{H}$에서 최적의 h를 선택하기 위한 우선순위 규칙 또는 가정의 집합이다.

예시:

가능한 가설:

$$h_1(x) = 2x + 1 \quad (\text{직선}) \tag{1}$$
$$h_2(x) = x^2 + x + 1 \quad (\text{포물선}) \tag{2}$$
$$h_3(x) = \sin(x) + 2 \quad (\text{사인 곡선}) \tag{3}$$
$$\vdots \tag{4}$$

선형회귀의 귀납적 편향: "h_1을 선택하자!" (직선만 고려)

귀납적 편향 없이는 학습 불가능!

정리 (Mitchell, 1980):

> 귀납적 편향이 없는 학습자는 본 적 없는 데이터에 대해 무작위 추측보다 나을 수 없다.

증명 (직관적):

1. 학습 데이터: $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
2. 새 입력: x_{new} (학습 중 본 적 없음)
3. 가능한 출력: $y_{new} \in \{0, 1\}$ (이진 분류라고 가정)

질문: y_{new}는 0일까요, 1일까요?

귀납적 편향 없으면:

- 0과 1 모두 **똑같이 가능**합니다
- 학습 데이터는 x_{new}에 대해 **아무 정보도 주지 않습니다**
- 동전 던지기와 같음: 정확도 50%

귀납적 편향 있으면:

- "가까운 점들은 비슷하다" → k-NN 사용 가능
- "직선 관계다" → 선형회귀로 보간 가능

정리: 귀납적 편향을 선택한다는 것

머신러닝 알고리즘을 선택하는 것 = 귀납적 편향을 선택하는 것

의미:

1. **선형회귀 선택** = "세상은 선형이다"라고 가정
2. **결정트리 선택** = "세상은 if-then 규칙이다"라고 가정
3. **신경망 선택** = "세상은 복잡하지만 매끄럽다"라고 가정

> 💡 **중요한 질문**
>
> "내 문제에 맞는 귀납적 편향은 무엇인가?"
>
> 이것이 바로 **도메인 지식**이 중요한 이유입니다!

1.2 공짜 점심은 없다 (No Free Lunch Theorem) 🍔

"만능 알고리즘은 없다"

머신러닝을 처음 배우는 사람들의 흔한 질문:

> "어떤 알고리즘이 가장 좋아요?"
>
> "무조건 딥러닝을 쓰면 되나요?"

답: 그런 건 없습니다! 이것이 바로 NFL 정리의 핵심입니다.

NFL 정리의 직관적 이해

비유: 운동화 선택하기 👟

질문: "가장 좋은 신발은 무엇인가요?"

답:
- 마라톤 할 때? → 러닝화 
- 농구 할 때? → 농구화 
- 수영 할 때? → 맨발 
- 등산 할 때? → 등산화 

모든 상황에서 최고인 신발은 없습니다!

머신러닝도 마찬가지입니다.

NFL 정리의 공식적 진술

Wolpert & Macready (1997):

> 모든 가능한 문제들에 대해 평균을 내면, 모든 학습 알고리즘의 성능은 동일하다.

수식적 표현:

모든 가능한 문제들에 대해 두 알고리즘의 성능을 합산하면:

> 💡 **수식 기호 설명**
>
> - $\sum_f$ (시그마): "모든 가능한 함수 f에 대해 더하기"
>   - 예: 함수가 f_1, f_2, f_3 세 개면 → 각각의 성능을 계산해 모두 더함
> - $E(h, f)$: **성능 함수(Performance Function)** 또는 **오차(Error)**
>   - "목표 함수 f가 주어졌을 때 가설 h의 평균 성능 또는 오차"
>   - 일반적으로 테스트 데이터에서의 **평균 손실값** 또는 **정확도**를 의미
> - $|$ (수직선): "~가 주어졌을 때"를 의미하는 **조건부 기호**
>   - 예: $P(y|x)$ → "입력 x가 주어졌을 때 출력 y의 확률"

$$\sum_f E(h_A, f) = \sum_f E(h_B, f)$$

여기서:

- h_A: 알고리즘 A가 학습한 가설
- h_B: 알고리즘 B가 학습한 가설
- D: 학습 데이터
- f: 문제 (목표 함수)

쉽게 풀어 읽으면:
"모든 가능한 문제들에 대해, 알고리즘 A가 틀린 횟수의 총합 = 알고리즘 B가 틀린 횟수의 총합"

구체적 비유:

```
문제 1 (이미지 분류): A가 잘함 (+10점)
문제 2 (소리 인식): B가 잘함 (+10점)
문제 3 (랜덤 데이터): 둘 다 못함 (0점)
...
문제 1000: ...

총합 = A: 500점, B: 500점 (같음!)
```

쉽게 말하면:

- 알고리즘 A가 어떤 문제에서 좋으면
- 다른 문제에서는 알고리즘 B가 더 좋을 것
- 평균적으로는 **모두 똑같음**

왜 이런 정리가 성립하는가?

증명의 핵심 아이디어:

1. **가능한 모든 문제를 고려**하면 (심지어 말도 안 되는 문제까지!)
2. 어떤 알고리즘이 잘 푸는 문제가 있으면
3. 그 알고리즘이 못 푸는 문제도 똑같이 많이 있음

NFL 정리의 수학적 증명 📐

> 💡 **증명을 읽기 전에**
>
> 이 증명은 조금 길지만, 단계별로 따라가면 이해할 수 있습니다.
> 먼저 직관적 예시를 보고, 그 다음 엄밀한 증명을 보세요!

준비: 기호와 정의

문제 설정:

- **입력 공간**: $\mathcal{X} = \{x_1, x_2, ..., x_m\}$ (유한 집합)
- **출력 공간**: $\mathcal{Y} = \{0, 1\}$ (이진 분류)
- **목표 함수**: $f : \mathcal{X} \to \mathcal{Y}$ (알고 싶은 진짜 답)
- **학습 데이터**: $D = \{(x_1, f(x_1)), ..., (x_n, f(x_n))\}$ (크기 $n < m$)

- **테스트 데이터**: $\mathcal{X} \setminus D$ (학습 중 안 본 데이터)

학습 알고리즘:

- 알고리즘 A: $D \mapsto h_A$ (학습 데이터를 받아서 가설 h_A를 출력)

성능 측정:

- **오차**: 테스트 데이터에서 틀린 개수

$$E(h, f) = \sum_{x \notin D} 1[h(x) \neq f(x)]$$

여기서 $1[\cdot]$는 지시 함수 (참이면 1, 거짓이면 0)

증명의 핵심 질문

> **질문**: 모든 가능한 목표 함수 f에 대해 평균을 내면,
> 알고리즘 A와 알고리즘 B의 성능이 같을까?

답: 그렇다! 이제 증명해봅시다.

증명 (Wolpert & Macready, 1997)

목표: 모든 학습 알고리즘 A, B에 대해:

$$\sum_f E(h_A, f) = \sum_f E(h_B, f)$$

Step 1: 가능한 목표 함수의 개수

$|\mathcal{X}| = m$이고 $|\mathcal{Y}| = 2$이면,

가능한 모든 함수 $f : \mathcal{X} \to \mathcal{Y}$의 개수는:

$$|\mathcal{F}| = 2^m$$

예시:

만약 $m = 3$이면 (입력이 3개), 가능한 함수: $2^3 = 8$개

$$f_1 : (0,0,0) \tag{5}$$
$$f_2 : (0,0,1) \tag{6}$$
$$f_3 : (0,1,0) \tag{7}$$
$$f_4 : (0,1,1) \tag{8}$$
$$f_5 : (1,0,0) \tag{9}$$
$$f_6 : (1,0,1) \tag{10}$$
$$f_7 : (1,1,0) \tag{11}$$
$$f_8 : (1,1,1) \tag{12}$$

Step 2: 학습 데이터와 일치하는 함수들

주어진 학습 데이터 D에 대해,

D와 일치하는 함수들의 집합:

$$\mathcal{F}_D = \{f \mid (x_i, f(x_i)) \in D\}$$

이 집합의 크기:

$$|\mathcal{F}_D| = 2^{m-n}$$

왜?

- 학습 데이터 n개는 이미 값이 정해짐
- 나머지 $m-n$개 입력에 대해서는 자유롭게 0 또는 1 선택

예시:

$m = 4, n = 2$

학습 데이터: $D = \{(x_1, 1), (x_2, 0)\}$

일치하는 함수들:

$$f_1 : x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0 \tag{13}$$
$$f_2 : x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1 \tag{14}$$
$$f_3 : x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0 \tag{15}$$
$$f_4 : x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1 \tag{16}$$

총 $2^{4-2} = 4$개

Step 3: 테스트 데이터의 모든 가능한 레이블 조합

테스트 데이터 크기: $m - n$

각 테스트 포인트는 0 또는 $1 \rightarrow$ 가능한 조합: 2^{m-n}개

핵심 관찰:

D와 일치하는 각 함수는, 테스트 데이터에 대해 **서로 다른 레이블 조합**을 가집니다!

즉, 테스트 데이터의 모든 가능한 레이블 패턴이 **똑같이 자주** 나타납니다.

Step 4: 알고리즘의 예측

알고리즘 A가 학습 데이터 D를 보고 예측 h_A를 만듭니다.

테스트 포인트 $x \notin D$에 대해:

- $h_A(x) = 0$ 또는 1로 예측

질문: 이 예측이 맞을 확률은?

Step 5: 정답의 분포

D와 일치하는 2^{m-n}개 함수 중:

- 절반 (2^{m-n-1}개)은 $f(x) = 0$
- 절반 (2^{m-n-1}개)은 $f(x) = 1$

왜?

테스트 데이터의 모든 레이블 조합이 똑같이 나타나므로!

Step 6: 알고리즘의 오차 계산

알고리즘 A가 $h_A(x) = 0$으로 예측했다고 가정.

틀리는 경우: $f(x) = 1$인 함수들

개수: 2^{m-n-1}개 (전체의 절반!)

모든 테스트 포인트에 대해 합산:

$$\sum_{f \in \mathcal{F}_D} E(h_A, f) = \sum_{f \in \mathcal{F}_D} \sum_{x \notin D} 1[h_A(x) \neq f(x)]$$

순서를 바꾸면:

$$= \sum_{x \notin D} \sum_{f \in \mathcal{F}_D} 1[h_A(x) \neq f(x)]$$

각 테스트 포인트 x에 대해:

- $h_A(x) = 0$이면, $f(x) = 1$인 함수가 2^{m-n-1}개
- $h_A(x) = 1$이면, $f(x) = 0$인 함수가 2^{m-n-1}개

어느 쪽이든 **틀리는 함수가 정확히 2^{m-n-1}개**!

Step 7: 전체 합산

테스트 포인트 개수: $m - n$

각 포인트마다 틀리는 함수: 2^{m-n-1}개

$$\sum_{f \in \mathcal{F}_D} E(h_A, f) = (m-n) \cdot 2^{m-n-1}$$

핵심:

이 값은 **알고리즘 A와 무관**합니다!

알고리즘 B도 똑같은 값을 얻습니다:

$$\sum_{f \in \mathcal{F}_D} E(h_B, f) = (m-n) \cdot 2^{m-n-1}$$

Step 8: 모든 학습 데이터에 대해 평균

위 결과를 모든 가능한 학습 데이터 D에 대해 합산하면:

$$\sum_{D} \sum_{f \in \mathcal{F}_D} E(h_A, f) = \sum_{D} \sum_{f \in \mathcal{F}_D} E(h_B, f)$$

결론:

$$\boxed{\sum_{f} E(h_A, f) = \sum_{f} E(h_B, f)}$$

모든 알고리즘의 평균 성능은 동일하다! ■

직관적 이해: 왜 이런 일이?

핵심 아이디어:

1. **테스트 데이터는 학습 중 안 봤다**
   - 학습 데이터 D는 테스트 포인트 x에 대해 정보를 주지 않음
2. **모든 레이블이 똑같이 가능하다**
   - $f(x) = 0$인 함수: 절반
   - $f(x) = 1$인 함수: 절반
3. **알고리즘이 뭘 예측하든 틀릴 확률 50%**
   - $h(x) = 0$ 예측 $\rightarrow$ $f(x) = 1$인 함수가 절반
   - $h(x) = 1$ 예측 $\rightarrow$ $f(x) = 0$인 함수가 절반
4. **모든 알고리즘이 똑같이 운이 나쁘다**
   - 알고리즘이 아무리 똑똑해도
   - "모든 가능한 함수"를 고려하면
   - 평균 정확도는 50% (동전 던지기)

구체적 예시로 검증

설정:

$$\text{입력:} \quad \mathcal{X} = \{x_1, x_2, x_3\} \tag{17}$$
$$\text{출력:} \quad \mathcal{Y} = \{0, 1\} \tag{18}$$
$$\text{학습 데이터:} \quad D = \{(x_1, 1)\} \tag{19}$$
$$\text{테스트 데이터:} \quad \{x_2, x_3\} \tag{20}$$

가능한 모든 함수 (8개):

함수	x_1	x_2	x_3	D와 일치?
f_1	1	0	0	✅
f_2	1	0	1	✅
f_3	1	1	0	✅
f_4	1	1	1	✅
f_5	0	0	0	❌
f_6	0	0	1	❌
f_7	0	1	0	❌
f_8	0	1	1	❌

D와 일치하는 함수: $\{f_1, f_2, f_3, f_4\}$ (4개 = 2^{3-1})

알고리즘 A의 예측:

$$h_A(x_2) = 0, \quad h_A(x_3) = 0$$

각 함수에서의 오차:

함수	$h_A(x_2)$ vs $f(x_2)$	$h_A(x_3)$ vs $f(x_3)$	총 오차
f_1	0 vs 0 ✅	0 vs 0 ✅	0
f_2	0 vs 0 ✅	0 vs 1 ❌	1
f_3	0 vs 1 ❌	0 vs 0 ✅	1
f_4	0 vs 1 ❌	0 vs 1 ❌	2

총 오차: $0 + 1 + 1 + 2 = 4$

알고리즘 B의 예측 (다르게):

$$h_B(x_2) = 1, \quad h_B(x_3) = 1$$

각 함수에서의 오차:

함수	$h_B(x_2)$ vs $f(x_2)$	$h_B(x_3)$ vs $f(x_3)$	총 오차
f_1	1 vs 0 ❌	1 vs 0 ❌	2
f_2	1 vs 0 ❌	1 vs 1 ✅	1
f_3	1 vs 1 ✅	1 vs 0 ❌	1
f_4	1 vs 1 ✅	1 vs 1 ✅	0

총 오차: $2 + 1 + 1 + 0 = 4$

결과: $\sum_f E(h_A, f) = \sum_f E(h_B, f) = 4$ ✅

검증: $(m - n) \cdot 2^{m-n-1} = 2 \cdot 2^{2-1} = 2 \cdot 2 = 4$ ✅

증명의 함의

1. 귀납적 편향의 필연성

NFL 정리가 말하는 것:

- "모든 문제"에서는 모든 알고리즘이 똑같음
- 하지만 "특정 문제"에서는 다름
- 따라서 **문제에 맞는 귀납적 편향을 선택**해야 함

2. 도메인 지식의 중요성

증명에서 핵심 가정:

- "모든 함수 f가 똑같이 가능하다"

현실:

- **어떤 함수는 더 자주 나타남** (예: 자연 법칙)
- **도메인 지식**으로 가능성 높은 함수에 집중해야 함

3. 실전에서는 NFL이 적용 안 됨

실제 세계의 문제들은:

- 무작위가 아님
- 패턴이 있음 (smoothness, locality)
- 따라서 **잘 설계된 알고리즘이 실제로 더 좋음**

NFL 정리를 넘어서

핵심 교훈:

$$\text{Good Algorithm} = \text{Problem Match} + \text{Inductive Bias}$$

1. **문제를 이해하라** (도메인 지식)
2. **적절한 귀납적 편향을 선택하라** (알고리즘 선택)
3. **경험적으로 검증하라** (교차 검증)

예시:

알고리즘 A: 선형 회귀

잘 푸는 문제:

$$y = 2x + 1 \tag{21}$$
$$y = -3x + 5 \tag{22}$$
$$y = 0.5x + 10 \tag{23}$$

못 푸는 문제:

$$y = x^2 \tag{24}$$
$$y = \sin(x) \tag{25}$$
$$y = \log(x) \tag{26}$$

알고리즘 B: 다항 회귀

잘 푸는 문제:

$$y = x^2 \tag{27}$$
$$y = x^3 + 2x^2 - 1 \tag{28}$$
$$y = (x-1)(x-2)(x-3) \tag{29}$$

못 푸는 문제:

$$y = \sin(x) \tag{30}$$
$$y = e^x \tag{31}$$
$$y = \text{불규칙한 함수} \tag{32}$$

모든 가능한 함수를 고려하면:

- 선형 회귀가 좋은 함수: 무한히 많음
- 다항 회귀가 좋은 함수: 무한히 많음
- 평균 성능: **똑같음!**

"그럼 어떤 알고리즘을 써야 하나요?"

답: 내 문제에 맞는 것을!

NFL 정리가 말해주는 것:

> ✅ **모든 문제**에 대해 평균 → 성능 같음
>
> ✅ **특정 문제**에 대해 → 성능 다름!

핵심:

1. **"모든 문제"는 현실적이지 않음**
2. **우리는 특정 문제만 관심 있음**
3. **내 문제의 특성을 파악하라!**

문제의 특성에 따른 알고리즘 선택

📏 데이터가 적을 때

좋은 선택: 선형 회귀, 로지스틱 회귀, 나이브 베이즈
이유: 강한 귀납적 편향 → 적은 데이터로도 학습

나쁜 선택: 딥러닝, 복잡한 신경망
이유: 많은 파라미터 → 과적합 위험

🗂️ 데이터가 많을 때

좋은 선택: 딥러닝, 랜덤 포레스트, XGBoost
이유: 복잡한 패턴 학습 가능

나쁜 선택: 너무 단순한 모델
이유: 데이터의 풍부한 정보를 못 활용 (과소적합)

📊 선형 관계가 예상될 때

좋은 선택: 선형 회귀, 로지스틱 회귀
이유: 귀납적 편향이 문제와 일치

나쁜 선택: 신경망
이유: 불필요하게 복잡함, 해석 어려움

🌀 비선형 관계가 예상될 때

좋은 선택: 신경망, 커널 SVM, 결정트리
이유: 복잡한 패턴 표현 가능

나쁜 선택: 선형 회귀
이유: 귀납적 편향이 문제와 안 맞음

🔍 해석 가능성이 중요할 때

```
좋은 선택: 결정트리, 선형 회귀, 로지스틱 회귀
이유: 왜 그런 예측을 했는지 설명 가능

나쁜 선택: 딥러닝, 랜덤 포레스트
이유: 블랙박스, 설명 어려움
```

NFL 정리와 귀납적 편향의 관계

연결:

1. **NFL 정리**: "만능 알고리즘은 없다"
2. **귀납적 편향**: "알고리즘마다 다른 가정을 한다"
3. **결론**: "문제에 맞는 가정을 가진 알고리즘을 선택하라"

수식적으로:

알고리즘의 성능은 **문제와 귀납적 편향의 일치도**에 비례합니다:

$$\text{Performance} \propto \text{Match}(\text{Problem}, \text{Inductive Bias})$$

예시:

문제 특성	최적 알고리즘	이유
선형 관계	선형 회귀	귀납적 편향 일치 ✅
선형 관계	신경망	불일치, 과도함 ❌
복잡한 이미지	CNN	지역적 패턴 가정 ✅
복잡한 이미지	선형 회귀	너무 단순 ❌
순차 데이터	RNN/LSTM	시간 의존성 가정 ✅
순차 데이터	k-NN	순서 무시 ❌

실전 교훈

1. 문제를 이해하라

질문:
- 데이터의 패턴은 선형인가, 비선형인가?
- 특징들 간의 관계는?
- 데이터가 충분한가?
- 해석 가능성이 필요한가?

2. 여러 알고리즘을 시도하라

전략:
1. 간단한 모델부터 시작 (선형 회귀, 로지스틱 회귀)
2. 성능 확인
3. 필요하면 복잡한 모델로 (트리, 신경망)
4. 교차 검증으로 비교

3. 도메인 지식을 활용하라

예시:
- 주식 예측: 시계열 모델 (ARIMA, LSTM)
- 이미지 인식: CNN
- 텍스트 분류: RNN, Transformer
- 표 형태 데이터: XGBoost, 랜덤 포레스트

"공짜 점심"이 있는 경우?

NFL 정리의 가정:

> "**모든** 가능한 문제"

현실:

> "우리가 실제로 만나는 문제들"

희소식:

실제 세계의 문제들은 **무작위가 아닙니다**!

실제 세계의 특성:
✅ 연속성 (smoothness): 비슷한 입력 → 비슷한 출력
✅ 규칙성 (regularity): 패턴이 존재함
✅ 지역성 (locality): 가까운 것끼리 영향을 줌

따라서:

- 이런 특성을 가정하는 알고리즘들은 **현실에서 잘 작동**합니다
- 완전 무작위 문제에서만 NFL 정리가 엄격히 적용됩니다

예시: 이미지 분류

CNN이 잘 작동하는 이유:

1. **지역성**: 가까운 픽셀끼리 관련 있음 (귀납적 편향)
2. **계층성**: 저수준 → 고수준 특징 (귀납적 편향)
3. **평행이동 불변성**: 위치가 달라도 같은 물체 (귀납적 편향)

이런 가정들이 **실제 이미지의 특성과 일치**하기 때문입니다!

1.3 학습의 철학적 기초 🎓

기계가 정말 "학습"하는가?

철학적 질문:

> 머신러닝 모델이 진짜로 "이해"하는 걸까, 아니면 그냥 "암기"하는 걸까?

이 질문은 간단하지 않습니다. 여러 관점에서 살펴봅시다.

관점 1: 행동주의 (Behaviorism)

주장:

> "이해"의 정의 = 올바른 행동을 하는 것

튜링 테스트 (Turing Test):

```
만약 기계가 사람처럼 행동하면,
그것을 "지능"이라고 불러도 된다.
```

머신러닝에 적용:

```
만약 모델이 정확한 예측을 하면,
그것을 "학습했다"고 불러도 된다.
```

비판:

```
반례: 중국어 방 (Chinese Room) 논증

사람이 규칙만 따라서 중국어 답변을 하면,
그 사람이 중국어를 "이해"한다고 할 수 있나?

마찬가지로, 모델이 패턴을 따라 예측하면,
그 모델이 데이터를 "이해"한다고 할 수 있나?
```

관점 2: 표현주의 (Representationalism)

주장:

> "학습" = 세계에 대한 내부 표현(representation)을 형성하는 것

머신러닝에서:

모델은 데이터의 **내부 표현**을 만듭니다.

예시: 이미지 분류 신경망

```
입력층:    픽셀 값 [0.2, 0.8, 0.3, ...]
          ↓
은닉층 1:  [0.1, 0.9, 0.0, ...] (선 검출기)
          ↓
은닉층 2:  [0.7, 0.3, 0.8, ...] (형태 검출기)
          ↓
은닉층 3:  [0.4, 0.6, 0.2, ...] (물체 부위 검출기)
          ↓
출력층:    [0.01, 0.95, 0.04] (고양이: 95%)
```

각 층은 점점 더 **추상적인 표현**을 만듭니다!

수식적으로:

$$x \to h_1(x) \to h_2(h_1(x)) \to ... \to y$$

여기서 h_i는 i번째 층의 표현 함수입니다.

관점 3: 압축 (Compression)

주장:

> "학습" = 데이터의 압축

정보이론적 관점:

학습이란 데이터에서 **규칙성을 추출**하여 **간결하게 표현**하는 것

예시:

데이터 (압축 전):

$$\text{점 } 1: \quad (1, 3) \tag{33}$$
$$\text{점 } 2: \quad (2, 5) \tag{34}$$
$$\text{점 } 3: \quad (3, 7) \tag{35}$$
$$\text{점 } 4: \quad (4, 9) \tag{36}$$
$$\text{점 } 5: \quad (5, 11) \tag{37}$$

저장 공간: 10개의 숫자 (5개 점 $\times$ 2개 좌표)

학습 (압축 후):

$$y = 2x + 1$$

저장 공간: 3개의 숫자 (기울기 2, 절편 1, 오차 0)

Kolmogorov 복잡도:

데이터의 "진짜 복잡도" = 데이터를 생성하는 가장 짧은 프로그램의 길이

$$K(x) = \min_{p: p \text{ outputs } x} |p|$$

학습의 목표:

$$\text{Learned Model} \approx \text{Shortest Description of Data}$$

관점 4: 일반화 (Generalization)

주장:

> "학습" = 일반화 능력을 갖추는 것

오컴의 면도날 (Occam's Razor):

> 같은 현상을 설명하는 여러 가설 중, 가장 단순한 것을 선택하라

왜?
단순한 모델이 **일반화를 더 잘** 하기 때문!

예시:

데이터:

$$\{(1, 2), (2, 4), (3, 6)\}$$

가능한 가설들:

가설 1 (단순):

$$y = 2x$$

- 파라미터: 1개 (기울기 2)
- 일반화: 좋음 ✅

가설 2 (복잡):

$$y = 2x + 0.00001(x - 1)(x - 2)(x - 3)$$

- 파라미터: 4개
- 데이터는 완벽히 맞음
- 일반화: 나쁨 ❌ (새 점에서 이상한 값 예측)

MDL (Minimum Description Length) 원리:

$$\text{Best Model} = \arg\min_{M}[\text{Length}(M) + \text{Length}(\text{Data}|M)]$$

- 모델 복잡도 + 모델로 설명 못하는 데이터
- 둘 사이의 **균형**을 찾아야 함!

학습의 단계: 암기 → 이해 → 일반화

머신러닝 모델이 학습하는 과정:

1단계: 암기 (Memorization)

```
특정 데이터 포인트를 그대로 기억
예: "고양이 사진 #1234 → 고양이"

일반화: 없음 ❌
```

2단계: 패턴 인식 (Pattern Recognition)

```
데이터에서 규칙성 발견
예: "뾰족한 귀 + 수염 → 고양이"

일반화: 보통 ✅
```

3단계: 추상화 (Abstraction)

```
높은 수준의 개념 형성
예: "고양이의 본질적 특징" 표현 학습

일반화: 좋음 ✅✅
```

수학적 표현:

학습 과정에서 손실함수의 변화:

$$\text{Training Error} \downarrow \text{ (암기 단계)} \tag{38}$$
$$\text{Validation Error} \downarrow \text{ (패턴 인식)} \tag{39}$$
$$\text{Test Error} \downarrow \text{ (일반화)} \tag{40}$$

귀납적 추론의 정당화 문제

David Hume의 문제 (1748):

> 과거가 미래를 보장하는가?
>
> "태양이 지난 1만년 동안 떴다고 해서, 내일도 뜰 것이라는 보장이 있나?"

머신러닝에 적용:

> "학습 데이터에서 패턴을 찾았다고 해서, 새 데이터에서도 그 패턴이 유효할 보장이 있나?"

답: 없습니다!

그런데 왜 작동하는가?

실용적 답변:

1. **세계의 규칙성**: 자연 법칙은 안정적
2. **IID 가정**: 데이터가 같은 분포에서 추출됨
3. **경험적 검증**: 실제로 잘 작동함

수식적 표현:

IID 가정 (Independent and Identically Distributed):

$$P(x_{\text{train}}, y_{\text{train}}) = P(x_{\text{test}}, y_{\text{test}})$$

이 가정이 깨지면 학습 실패!

예: 분포 변화 (Distribution Shift)

```
학습 데이터: 여름철 아이스크림 판매량
테스트 데이터: 겨울철 아이스크림 판매량

IID 가정: 위반 ❌
결과: 예측 실패
```

PAC 학습 이론 (Probably Approximately Correct)

Valiant (1984):

귀납적 학습을 수학적으로 정당화하려는 시도

질문:

> "학습 알고리즘이 '충분히 좋은' 가설을 '높은 확률로' 찾을 수 있을까?"

PAC 학습 가능의 정의:

알고리즘 A가 개념 클래스 C를 PAC 학습한다 ⇔

모든 $\epsilon > 0, \delta > 0$에 대해, 충분히 많은 샘플 m이 있어서:

$$P(\text{error}(h) \leq \epsilon) \geq 1 - \delta$$

쉽게 말하면:

- ϵ: 허용 오차 ("충분히 좋은")
- δ: 실패 확률 ("높은 확률로")
- m: 필요한 데이터 개수

예시:

$\epsilon = 0.05$ (오차 5% 이하)

$\delta = 0.01$ (99% 확률로)

질문: 몇 개의 데이터가 필요한가?

답:

$$m \geq \frac{1}{\epsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

의미:

- 데이터가 충분하면, 좋은 가설을 높은 확률로 찾을 수 있다는 **수학적 보장**!

정리: 학습의 철학적 정당화

1. 실용적 정당화

✅ 경험적으로 작동함
✅ 많은 실제 문제 해결
✅ 예측 성능으로 검증 가능

2. 수학적 정당화

✅ PAC 학습 이론
✅ VC 차원 (다음 챕터)
✅ 통계적 학습 이론

3. 철학적 한계 인정

⚠️ 귀납의 문제는 완전히 해결 불가
⚠️ IID 가정이 필요
⚠️ 분포 변화에 취약

결론:

머신러닝은 **실용적으로 유용**하지만, **철학적으로 완벽하지는 않음**

우리가 할 수 있는 최선:

1. 가정을 명확히 하기 (귀납적 편향)
2. 검증 철저히 하기 (교차 검증, 테스트)
3. 한계를 인정하기 (분포 변화, 적대적 예시)

1장 요약 정리 📝

핵심 개념

1. 귀납적 편향 (Inductive Bias)

```
정의: 학습 알고리즘이 본 적 없는 데이터를 예측하기 위해 사용하는 사전 가정
역할: 일반화를 가능하게 함
중요성: 귀납적 편향 없이는 학습 불가능
```

2. 공짜 점심은 없다 (No Free Lunch)

```
의미: 모든 문제에 최고인 알고리즘은 없음
교훈: 문제 특성에 맞는 알고리즘 선택이 중요
실천: 도메인 지식 활용, 여러 모델 비교
```

3. 학습의 정당화

```
귀납의 문제: 과거→미래 추론은 논리적으로 불완전
실용적 해결: IID 가정, 경험적 검증
수학적 틀: PAC 학습 이론
```

알고리즘 선택 가이드

상황	추천 알고리즘	이유
데이터 적음	선형 회귀, 로지스틱 회귀	강한 귀납적 편향
데이터 많음	딥러닝, XGBoost	복잡한 패턴 학습

상황	추천 알고리즘	이유
선형 관계	선형 모델	가정과 문제 일치
비선형 관계	트리, 신경망, SVM	유연성
해석 필요	결정트리, 선형 모델	투명성
이미지	CNN	지역적 패턴
텍스트	RNN, Transformer	순차적 패턴
시계열	ARIMA, LSTM	시간 의존성

핵심 질문과 답

Q1: 왜 머신러닝이 작동하는가?

A: 귀납적 편향 덕분.
"세상은 규칙적이다"라는 가정이 실제로 맞기 때문.

Q2: 가장 좋은 알고리즘은?

A: 없음 (NFL 정리).
문제마다 최적 알고리즘이 다름.

Q3: 어떻게 알고리즘을 선택하나?

A: 1) 문제 특성 파악
2) 도메인 지식 활용
3) 여러 모델 비교 (교차 검증)

Q4: 학습이 실패하는 경우는?

A: 1) 귀납적 편향이 문제와 안 맞을 때
2) IID 가정이 깨질 때 (분포 변화)
3) 데이터가 너무 적을 때

다음 챕터 예고 

Chapter 2에서는:

- **어떻게** 학습하는가?
- 손실함수: 얼마나 틀렸는지 측정
- 최적화: 최선의 모델 찾기
- 경사하강법: 산을 내려가는 방법

연결:

```
Chapter 1: 왜 학습이 작동하는가? (철학적 기초)
         ↓
Chapter 2: 어떻게 학습하는가? (수학적 메커니즘)
```

🎓 복습 문제

1. k-NN의 귀납적 편향은 무엇인가?
2. NFL 정리가 "딥러닝이 최고다"라는 주장을 반박하는 이유는?
3. PAC 학습에서 ϵ과 δ의 의미는?
4. 분포 변화(distribution shift)가 왜 위험한가?

Chapter 2: 어떻게 학습하는가? 🎓

들어가며

Chapter 1에서 우리는 **왜** 머신러닝이 작동하는지 철학적, 수학적 기초를 다졌습니다. 이제 **어떻게** 학습이 일어나는지 구체적인 메커니즘을 이해할 차례입니다.

머신러닝 학습 과정의 핵심 질문:

> 🤔 "모델이 '좋다' 또는 '나쁘다'는 걸 어떻게 측정하지?"
>
> 🤔 "수많은 가능한 모델 중에서 최선을 어떻게 찾지?"
>
> 🤔 "컴퓨터가 스스로 개선하는 과정은 무엇인가?"

이 질문들에 답하기 위해 세 가지 핵심 개념을 배웁니다:

1. 손실함수 (Loss Function): 얼마나 틀렸는지 측정
2. 최적화 (Optimization): 최선의 해를 찾기
3. 경사하강법 (Gradient Descent): 최적화를 실제로 수행하는 알고리즘

2.1 손실함수 (Loss Function) 📉

손실함수란?

> 📘 **손실함수 (Loss Function)**
>
> 모델의 예측이 **얼마나 틀렸는지**를 수치화하는 함수

직관적 이해:

손실함수는 모델의 "성적표"입니다.

- 점수가 **낮을수록** 좋음 (틀린 정도를 측정하므로)
- 완벽한 예측: 손실 = 0
- 엉망인 예측: 손실 = 매우 큰 값

생활 속 비유 🎯

상황: 다트 게임

```
목표: 과녁 중심(정답)에 맞추기
실제: 다트가 어딘가에 꽂힘(예측)

손실함수 = 과녁 중심에서 떨어진 거리
```

수식으로:

$$\text{Loss} = \text{Distance}(\text{Target}, \text{Prediction})$$

- 중심에 명중: 거리 = 0 (완벽!)
- 과녁 밖: 거리 = 크다 (나쁨!)

수학적 정의

일반적으로 손실함수는 다음과 같이 정의됩니다:

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$$

> 💡 **기호 설명**
>
> - L: 손실함수 (Loss function)
> - $\mathcal{Y}$: 출력 공간 (가능한 모든 출력값의 집합)
>   - 예: 회귀면 $\mathcal{Y} = \mathbb{R}$ (모든 실수)
>   - 예: 이진 분류면 $\mathcal{Y} = \{0, 1\}$
> - $\times$: 곱집합 (두 개의 입력을 받는다는 의미)
> - $\to$: "~로 매핑된다"
> - $\mathbb{R}^+$: 음이 아닌 실수 (0, 0.5, 1, 100, ...)
>
> **쉽게 말하면:**
>
> "손실함수는 (실제값, 예측값) 두 개를 받아서 → 0 이상의 숫자를 돌려준다"

구체적으로:

하나의 데이터 포인트 (x, y)에 대해:

$$L(y, \hat{y}) = L(y, f(x))$$

> **기호 의미:**
>
> - y: 실제 정답 (ground truth, 레이블)
>   - 예: 집값 3억, 레이블 "고양이"
> - $\hat{y}$ (y hat): 모델의 예측
>   - "^" (hat)은 "추정값" 또는 "예측값"을 의미
>   - 예: 예측 2.8억, 예측 "개"
> - $f(x)$: 모델이 입력 x를 받아서 출력한 값 = $\hat{y}$
> - $L(y, \hat{y})$: 정답과 예측 사이의 차이를 수치화한 손실

전체 데이터셋에 대한 손실:

$$\mathcal{L}(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))$$

> **쉽게 말하면:**

> "모든 데이터에 대한 손실의 평균"
>
> ```
> 데이터 1의 손실: 0.5
> 데이터 2의 손실: 0.3
> 데이터 3의 손실: 0.8
>
> 평균 손실 = (0.5 + 0.3 + 0.8) / 3 = 0.53
> ```

이를 **경험적 손실 (Empirical Loss)** 또는 **경험적 위험 (Empirical Risk)**이라고 합니다.

왜 손실함수가 필요한가?

1. 객관적 평가

"좋다" 또는 "나쁘다"는 주관적입니다. 손실함수는 이를 **수치**로 만듭니다.

```
"이 모델이 더 좋은 것 같아요" ❌ (주관적)
"이 모델의 손실이 0.23이고, 저 모델은 0.45입니다" ✅ (객관적)
```

2. 최적화 가능

수치가 있어야 "더 좋게" 만들 수 있습니다.

$$\text{목표: } \min_{f} \mathcal{L}(f)$$

3. 자동 학습

컴퓨터가 손실을 줄이는 방향으로 자동으로 개선할 수 있습니다.

회귀 문제의 손실함수

회귀는 **연속적인 값**을 예측하는 문제입니다.

1 평균 제곱 오차 (Mean Squared Error, MSE)

정의:

$$L_{\text{MSE}}(y, \hat{y}) = (y - \hat{y})^2$$

> 💡 **수식 풀이**
>
> - $(y - \hat{y})$: 오차 (실제값 - 예측값)

    - 양수면: 예측이 작음 (과소 예측)
    - 음수면: 예측이 큼 (과대 예측)
- $(y - \hat{y})^2$: 오차를 제곱
    - **왜 제곱?**
        a. 음수 오차도 양수로 (|-2| = 4)
        b. 큰 오차에 더 큰 페널티 (오차 2 → 4, 오차 10 → 100)

구체적 계산:

```
실제: y = 5, 예측: ŷ = 3
오차: 5 - 3 = 2
MSE: 2² = 4

실제: y = 5, 예측: ŷ = 7
오차: 5 - 7 = -2
MSE: (-2)² = 4  (음수여도 4!)
```

전체 데이터셋:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

쉽게 말하면:

"모든 데이터의 (오차)²를 평균낸 것"

직관:

- 오차 $(y - \hat{y})$를 제곱
- 큰 오차에 **페널티**를 많이 줌

예시: 집값 예측

실제 집값 (y)	예측 ($\hat{y}$)	오차	제곱 오차
3억	2.8억	-0.2억	0.04억²
5억	5.3억	+0.3억	0.09억²
4억	3.9억	-0.1억	0.01억²

$$\text{MSE} = \frac{0.04 + 0.09 + 0.01}{3} = 0.047\text{억}^2$$

장점:

- ✅ 미분 가능 (경사하강법 사용 가능)
- ✅ 수학적으로 다루기 쉬움
- ✅ 볼록 함수 (convex) - 지역 최소값 = 전역 최소값

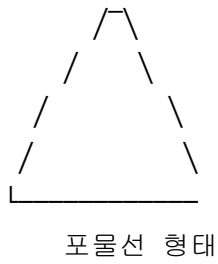
단점:

- ❌ 이상치(outlier)에 민감
- ❌ 큰 오차를 과도하게 페널티

시각화:

```
y=3일 때, 예측값에 따른 손실:

ŷ:    1    2    3    4    5
Loss: 4    1    0    1    4
```



포물선 형태

2 평균 절대 오차 (Mean Absolute Error, MAE)

정의:

$$L_{\text{MAE}}(y, \hat{y}) = |y - \hat{y}|$$

전체 데이터셋:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

직관:

- 오차의 절댓값
- 모든 오차를 **균등하게** 취급

같은 예시로 계산:

실제 집값 (y)	예측 ($\hat{y}$)	오차	절대 오차
3억	2.8억	-0.2억	0.2억
5억	5.3억	+0.3억	0.3억
4억	3.9억	-0.1억	0.1억

$$\text{MAE} = \frac{0.2 + 0.3 + 0.1}{3} = 0.2\text{억}$$

장점:

- ✅ 이상치에 덜 민감
- ✅ 해석이 직관적 (평균적으로 얼마나 틀렸는지)

단점:

- ❌ 0에서 미분 불가능 (최적화가 약간 까다로움)

MSE vs MAE 비교:

```
이상치가 있는 경우:

데이터: [1, 2, 3, 100] (100이 이상치)
예측:   [1, 2, 3, 3]

MSE = (0² + 0² + 0² + 97²) / 4 = 2352.25  (큰 페널티!)
MAE = (0 + 0 + 0 + 97) / 4 = 24.25        (상대적으로 작음)
```

3 Huber Loss

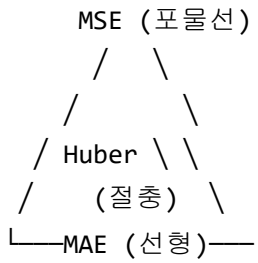
MSE와 MAE의 **절충안**:

$$L_{\text{Huber}}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta \cdot |y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

직관:

- 작은 오차: MSE처럼 동작 (부드러운 최적화)
- 큰 오차: MAE처럼 동작 (이상치에 강건)

시각화:



분류 문제의 손실함수

분류는 **이산적인 클래스**를 예측하는 문제입니다.

1 0-1 손실 (Zero-One Loss)

정의:

$$L_{0-1}(y, \hat{y}) = 1[y \neq \hat{y}] = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases}$$

직관:

- 맞으면 0점 (좋음)
- 틀리면 1점 (나쁨)

예시:

실제 (y)	예측 ($\hat{y}$)	손실
고양이	고양이	0
고양이	개	1
개	개	0

$$\text{평균 손실} = \frac{0 + 1 + 0}{3} = 0.333$$

문제점:

- ❌ **미분 불가능** (경사하강법 사용 불가)
- ❌ 예측의 **확신도**를 반영 못함

```
예측 A: "99% 고양이"인데 틀림 → 손실 = 1
예측 B: "51% 고양이"인데 틀림 → 손실 = 1

둘 다 같은 페널티? 불공평!
```

2 교차 엔트로피 손실 (Cross-Entropy Loss)

왜 로그(log)를 사용하는가?

직관: **놀람의 정도**를 측정하기 위해!

생활 속 비유:

```
내일 비 올 확률 예측:

날씨앱: "비 올 확률 90%"
실제: 비가 옴
당신의 반응: "그럴 줄 알았어" (놀람 ↓ = 손실 ↓)

날씨앱: "비 올 확률 1%"
실제: 비가 옴
당신의 반응: "헐! 완전 예상 못했는데!" 😱 (놀람 ↑↑ = 손실 ↑↑)
```

수학적으로:

- 확률 90% → $-\log(0.9) = 0.11$ (작은 손실)
- 확률 1% → $-\log(0.01) = 4.6$ (큰 손실)

로그는 **확률이 낮을수록 기하급수적으로 큰 페널티**를 줍니다!

이진 분류 (Binary Classification):

$$L_{\text{CE}}(y, \hat{y}) = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

> 💡 **수식 풀이**
>
> **기호:**
>
> - $y \in \{0, 1\}$: 실제 레이블 (0 또는 1)
> - $\hat{y} \in [0, 1]$: 예측 확률 (0~1 사이의 실수)
> - $\log$: 자연로그 (밑이 e인 로그)
>
> **왜 이런 식인가?**

> 경우를 나눠봅시다:
>
> **Case 1: 실제 y = 1 (정답이 1)**
>
> ```
> L = -[1·log(ŷ) + 0·log(1-ŷ)]
>   = -log(ŷ)
>
> ŷ = 0.9 → L = -log(0.9) = 0.11  (확신하고 맞춤, 좋음!)
> ŷ = 0.5 → L = -log(0.5) = 0.69  (불확실)
> ŷ = 0.1 → L = -log(0.1) = 2.30  (확신하고 틀림, 나쁨!)
> ```
>
> **Case 2: 실제 y = 0 (정답이 0)**
>
> ```
> L = -[0·log(ŷ) + 1·log(1-ŷ)]
>   = -log(1-ŷ)
>
> ŷ = 0.1 → L = -log(0.9) = 0.11  (확신하고 맞춤, 좋음!)
> ŷ = 0.5 → L = -log(0.5) = 0.69  (불확실)
> ŷ = 0.9 → L = -log(0.1) = 2.30  (확신하고 틀림, 나쁨!)
> ```
>
> **핵심:**
>
> - 정답에 대한 확률이 높을수록 → 손실 작음 ✅
> - 정답에 대한 확률이 낮을수록 → 손실 큼 ❌

직관:

확률적 예측의 **불확실성**을 측정합니다.

예시: 스팸 분류

실제: 스팸 ($y = 1$)

예측 확률 ($\hat{y}$)	손실
0.99 (매우 확신)	$-\log(0.99) = 0.01$ ✅
0.70 (약간 확신)	$-\log(0.70) = 0.36$
0.51 (거의 찍음)	$-\log(0.51) = 0.67$
0.01 (완전 틀림)	$-\log(0.01) = 4.61$ ❌

관찰:

- 확신하고 맞추면: 손실 ≈ 0
- 확신하고 틀리면: 손실 → ∞

다중 분류 (Multi-class Classification):

K개 클래스가 있을 때:

$$L_{\text{CE}}(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

여기서:

- y: 원-핫 인코딩 벡터 (예: $[0, 1, 0]$)
- $\hat{y}$: 예측 확률 분포 (예: $[0.2, 0.7, 0.1]$)

예시: 동물 분류

실제: 고양이 → $y = [0, 1, 0]$ (개, 고양이, 새)

예측: $\hat{y} = [0.1, 0.8, 0.1]$

$$L_{\text{CE}} = -[0 \cdot \log(0.1) + 1 \cdot \log(0.8) + 0 \cdot \log(0.1)] = -\log(0.8) = 0.22$$

장점:

- ✅ 미분 가능
- ✅ 확률적 예측에 적합
- ✅ 확신도를 반영

3 Hinge Loss (SVM용)

정의:

$$L_{\text{Hinge}}(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$$

여기서 $y \in \{-1, +1\}$, $\hat{y} \in \mathbb{R}$

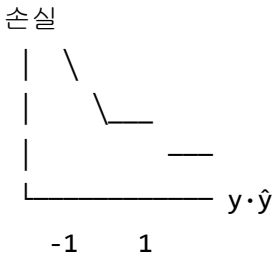
직관:

"마진"을 최대화합니다.

정답 클래스: +1

ŷ > +1: 손실 = 0 (확실히 맞춤)
ŷ = +1: 손실 = 0 (경계선)
0 < ŷ < +1: 손실 > 0 (맞추긴 했지만 확신 부족)
ŷ < 0: 손실 큼 (틀림)

시각화:



마진 1 이상이면 손실 0

손실함수 선택 가이드

문제 유형	추천 손실함수	이유
회귀 (일반)	MSE	미분 가능, 최적화 쉬움
회귀 (이상치 多)	MAE, Huber	이상치에 강건
이진 분류	Cross-Entropy	확률 해석, 미분 가능
다중 분류	Cross-Entropy	표준, 효과적
SVM	Hinge Loss	마진 최대화

손실함수의 성질

좋은 손실함수의 조건:

1. **미분 가능 (Differentiable)**: 경사하강법 사용 가능
2. **볼록 (Convex)**: 지역 최소값 = 전역 최소값
3. **해석 가능 (Interpretable)**: 값의 의미가 명확
4. **문제에 적합 (Task-appropriate)**: 목표와 일치

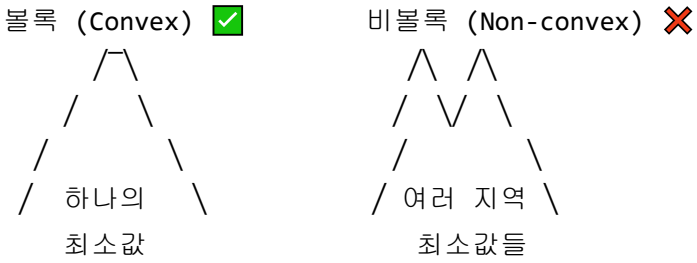
수학적으로:

손실함수 L이 **볼록**이면:

$$L(\lambda x_1 + (1-\lambda)x_2) \leq \lambda L(x_1) + (1-\lambda)L(x_2)$$

모든 x_1, x_2와 $\lambda \in [0, 1]$에 대해.

시각화:



2.2 최적화 (Optimization) 🔍

최적화란?

> 📘 **최적화 (Optimization)**
>
> 목적함수(손실함수)를 **최소화**(또는 최대화)하는 파라미터를 찾는 과정

수학적 정의:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta)$$

> 💡 **수식이 어렵다면?**
>
> 이 수식을 한국어로 번역하면:
>
> **θ^*는 $\mathcal{L}(\theta)$를 최소화하는 θ 값이다**
>
> - `arg` : argument의 약자 = "인자", "값"
> - `min` : minimum의 약자 = "최소값을 만드는"
> - `argmin` : "최소값을 만드는 값"
>
> **구체적 예시:**

```
함수: f(x) = (x-3)²

min f(x) = 0          ← 최소값 자체
argmin f(x) = 3       ← 최소값을 만드는 x 값

우리가 찾고 싶은 건 argmin! (어떤 값일 때 최소인지)
```

기호 설명:

- θ (세타): 모델의 파라미터 (가중치, 편향 등)
  - 예: 선형회귀에서 $\theta = [w, b]$
- $\mathcal{L}(\theta)$: 파라미터가 θ일 때의 손실
  - 예: $\mathcal{L}(w = 2, b = 1) = 0.5$
- θ^* (세타 스타): **최적** 파라미터 (손실이 가장 작을 때)
  - 우리가 찾고 싶은 답!

생활 속 비유 ⛰️

상황: 산 정상에서 출발해 가장 낮은 골짜기를 찾기

```
현재 위치 = 현재 파라미터
높이 = 손실 (낮을수록 좋음)
목표 = 가장 낮은 지점 찾기

전략:
1. 주변을 둘러본다 (기울기 계산)
2. 가장 낮은 방향으로 한 걸음 내려간다 (파라미터 업데이트)
3. 반복 (더 이상 내려갈 곳이 없을 때까지)
```

왜 최적화가 필요한가?

문제: 파라미터 공간이 너무 크다

간단한 신경망도:

- 입력: 100개
- 은닉층: 50개
- 출력: 10개

파라미터 개수:

$$(100 \times 50) + 50 + (50 \times 10) + 10 = 5,560\text{개}$$

모든 조합을 시도?

각 파라미터를 10개 값만 시도해도:

$$10^{5560}\text{가지 조합}$$

우주의 원자 개수($\approx 10^{80}$)보다 훨씬 많습니다! 😱

해결책: 똑똑하게 탐색하는 **최적화 알고리즘**

최적화 문제의 유형

1 제약 없는 최적화 (Unconstrained Optimization)

$$\min_{\theta} \mathcal{L}(\theta)$$

파라미터에 제약 조건 없음.

예시: 선형 회귀

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} (y_i - (wx_i + b))^2$$

w, b는 어떤 값이든 가능.

2 제약 있는 최적화 (Constrained Optimization)

$$\begin{aligned} \min_{\theta} \quad & \mathcal{L}(\theta) && (41) \\ \text{subject to} \quad & g(\theta) \leq 0 && (42) \\ & h(\theta) = 0 && (43) \end{aligned}$$

예시: 가중치에 제약

$$\min_{w} \mathcal{L}(w) \quad \text{subject to} \quad ||w||_2 \leq C$$

가중치의 크기가 C를 넘으면 안 됨.

최적화의 필요조건: 미분

1차 필요조건 (First-order Necessary Condition):

θ^*가 최소값이면:

$$\nabla \mathcal{L}(\theta^*) = 0$$

직관:

```
기울기 = 0 ↔ 평평한 곳 ↔ 극값 후보

  /‾\      기울기가 0인 지점
 /   \
      ← 최소값
```

그래디언트 (Gradient):

$$\nabla \mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1} \\ \frac{\partial \mathcal{L}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_d} \end{bmatrix}$$

> 💡 **편미분이 뭔가요?** (미분을 아는 사람도 헷갈림!)
>
> **비유: 산의 기울기 측정**
>
> 상황: 당신은 산 중턱에 서 있음
> 질문: "어느 방향이 가장 가파른가?"
>
> 2차원 땅 (x, y 좌표):
>
> - 북쪽(y 방향)으로 가면? → $\frac{\partial \text{높이}}{\partial y}$
> - 동쪽(x 방향)으로 가면? → $\frac{\partial \text{높이}}{\partial x}$
>
> **편미분 = "한 방향으로만" 기울기 측정**
> (다른 방향은 움직이지 않고 고정)
>
> **구체적 예시:**

```
함수: h(x, y) = x² + y²  (그릇 모양)
현재 위치: (x=2, y=3)

∂h/∂x = 2x = 2(2) = 4
의미: "x를 1 증가하면 높이가 약 4 증가"
       (이때 y=3은 고정!)

∂h/∂y = 2y = 2(3) = 6
의미: "y를 1 증가하면 높이가 약 6 증가"
       (이때 x=2는 고정!)
```

기호 설명

- ∇ (nabla, 나블라): 그래디언트 기호
  - "모든 방향으로의 기울기를 모아놓은 것"
- $\frac{\partial}{\partial \theta_i}$ (편미분 기호 ∂): θ_i에 대한 미분
  - 다른 변수는 고정하고 θ_i만 변화시킬 때의 변화율
- d: 파라미터의 개수

쉽게 말하면:

그래디언트는 **각 파라미터를 조금 바꿨을 때 손실이 얼마나 변하는지**를 나타내는 벡터입니다.

구체적 예시:

```
파라미터: θ = [w, b]
손실: L(w, b) = (w-2)² + (b-1)²

그래디언트:
▽L = [∂L/∂w, ∂L/∂b] = [2(w-2), 2(b-1)]

w=0, b=0일 때:
▽L = [2(0-2), 2(0-1)] = [-4, -2]

의미:
- w를 +1 증가 → 손실이 약 -4만큼 변함 (감소!)
- b를 +1 증가 → 손실이 약 -2만큼 변함 (감소!)
```

2차 충분조건 (Second-order Sufficient Condition):

왜 1차 미분만으로는 부족한가?

문제 상황:

```
함수: f(x) = -x³

x=0에서:
- 1차 미분: f'(0) = 0 ✅ (평평함)
- 최소값? ❌ (사실 변곡점!)

      /
    /
  /      ← 변곡점 (최소도 최대도 아님)
```

1차 미분만으로는 최소/최대/변곡점 구분 불가!

해결: 2차 미분 필요

비유: 자동차 운전

- **1차 미분 (속도)**: "지금 얼마나 빠른가"
- **2차 미분 (가속도)**: "속도가 증가 중인가 감소 중인가"

최소값 조건:

1. $f'(x) = 0$ (평평)
2. $f''(x) > 0$ (아래로 볼록 = 그릇 모양)

θ^*가 최소값이려면:

1. $\nabla \mathcal{L}(\theta^*) = 0$ (1차 조건)
2. $\nabla^2 \mathcal{L}(\theta^*)$ 가 양정부호 (positive definite)

헤시안 (Hessian) 행렬:

$$\nabla^2 \mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial \theta_1^2} & \frac{\partial^2 \mathcal{L}}{\partial \theta_1 \partial \theta_2} & \cdots \\ \frac{\partial^2 \mathcal{L}}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 \mathcal{L}}{\partial \theta_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

> 💡 **헤시안이 뭔가요?**
>
> - $\frac{\partial^2}{\partial \theta_i^2}$: **2차 편미분** = 기울기의 기울기
>   - "얼마나 빠르게 기울기가 변하는가"
> - 헤시안 = **2차 미분들을 모아놓은 행렬**

> **쉬운 비유:**
>
> ```
> 1차 미분 (그래디언트):
>   "지금 이 언덕이 얼마나 가파른가?" (속도)
>
> 2차 미분 (헤시안):
>   "언덕의 가파름이 얼마나 빠르게 변하는가?" (가속도)
> ```
>
> **구체적 예시:**
>
> ```
> 함수: f(x) = x²
>
> 1차 미분: f'(x) = 2x      (기울기)
> 2차 미분: f''(x) = 2      (곡률, 항상 양수)
>
> → 2차 미분이 양수 = 위로 볼록 = 최소값 가능 ✅
> ```

직관:

헤시안은 **곡률**(얼마나 휘어졌는지)을 나타냅니다.

- **양정부호** (positive definite): 모든 방향으로 위로 휨

  ```
    /‾\    ← 그릇 모양
  /    \
  ```

  → 최소값 ✅
- **음정부호** (negative definite): 모든 방향으로 아래로 휨

  ```
  \    /    ← 산봉우리 모양
    \_/
  ```

  → 최대값
- **부정부호** (indefinite): 어떤 방향은 위, 어떤 방향은 아래

  ```
    /\      ← 안장 모양
  _/  \_
  ```

  → 안장점 (saddle point)

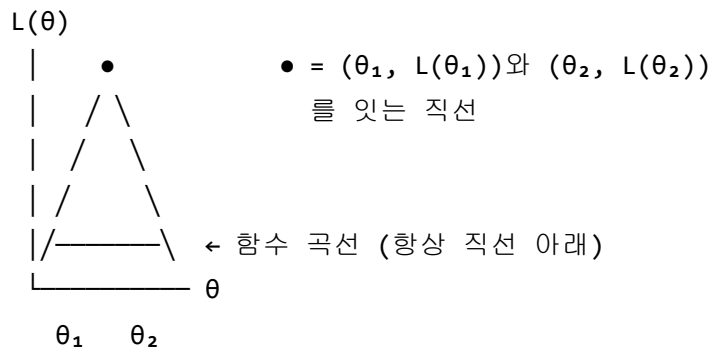
볼록 최적화 (Convex Optimization)

정의:

손실함수 $\mathcal{L}$이 **볼록**이면:

$$\mathcal{L}(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda\mathcal{L}(\theta_1) + (1-\lambda)\mathcal{L}(\theta_2)$$

모든 θ_1, θ_2와 $\lambda \in [0,1]$에 대해.

> 💡 **이 수식이 무슨 뜻인가요?**
>
> **기호 먼저 이해:**
>
> - θ_1, θ_2: 파라미터 공간의 임의의 두 점
> - $\lambda \in [0,1]$: 0과 1 사이의 실수 (예: 0.3, 0.7)
> - $\lambda\theta_1 + (1-\lambda)\theta_2$: 두 점 사이의 **어떤 점**
>   - $\lambda = 0$이면 $\rightarrow \theta_2$
>   - $\lambda = 1$이면 $\rightarrow \theta_1$
>   - $\lambda = 0.5$면 $\rightarrow$ 정확히 중간점
>
> **기하학적 의미:**
>
> 
>
>
> **쉽게 말하면:**
>
> "두 점을 직선으로 이으면, 그 직선은 항상 함수 그래프보다 위에 있다"
>
> **구체적 예시:**

```
θ₁ = 0, θ₂ = 4, λ = 0.5 (중간점)
함수: L(θ) = θ²

왼쪽 (중간점의 함수값):
  L(0.5×0 + 0.5×4) = L(2) = 4

오른쪽 (함수값들의 평균):
  0.5×L(0) + 0.5×L(4) = 0.5×0 + 0.5×16 = 8

4 ≤ 8 ✅ (볼록!)
```

황금률:

> 볼록 함수에서는 **지역 최소값 = 전역 최소값**

증명 스케치:

만약 θ_1이 지역 최소값인데 전역 최소값 θ^*가 따로 있다면:

$$\mathcal{L}(\theta^*) < \mathcal{L}(\theta_1)$$

θ_1과 θ^* 사이의 점 $\theta_\lambda = \lambda\theta^* + (1-\lambda)\theta_1$을 보면:

볼록성에 의해:

$$\mathcal{L}(\theta_\lambda) \leq \lambda\mathcal{L}(\theta^*) + (1-\lambda)\mathcal{L}(\theta_1) < \mathcal{L}(\theta_1)$$

이는 θ_1이 지역 최소값이라는 가정에 모순! ■

결론:

볼록 최적화는 **항상 전역 최적해를 찾을 수 있습니다** ✅

볼록한 손실함수 예시:

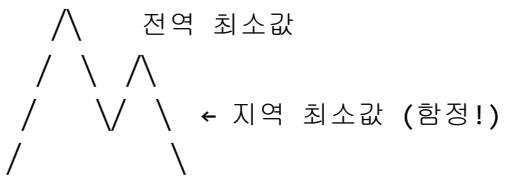
- MSE (선형 회귀)
- Cross-Entropy (로지스틱 회귀)
- Hinge Loss (SVM)

비볼록 손실함수 예시:

- 신경망의 손실함수 (일반적으로)

비볼록 최적화의 어려움

신경망 같은 비볼록 문제에서는:



문제점:

1. **지역 최소값**: 주변보다 낮지만 전역 최소값은 아님
2. **안장점 (Saddle Point)**: 어떤 방향은 최소, 다른 방향은 최대
3. **평평한 영역 (Plateau)**: 기울기가 거의 0인 넓은 영역

실전에서의 대처:

- 무작위 초기화 (여러 번 시도)
- 모멘텀 (관성 이용)
- 적응적 학습률 (Adam 등)

2.3 경사하강법 (Gradient Descent) ⬇️

경사하강법이란?

> 📘 **경사하강법 (Gradient Descent)**
>
> 손실함수의 **기울기(경사)** 를 이용하여 반복적으로 파라미터를 업데이트하는 최적화 알고리즘

핵심 아이디어:

"가장 가파르게 내려가는 방향으로 한 걸음씩 이동"

산 내려가기 비유 ⛰️

상황: 짙은 안개 속에서 산을 내려가기

시야: 발 주변만 보임 (지역적 정보)
도구: 경사계 (기울기 측정)

전략:
1. 현재 위치에서 가장 가파른 방향 찾기
2. 그 방향으로 한 걸음 내려가기
3. 반복

비유와 수학의 대응:

비유	수학
현재 위치	$\theta^{(t)}$
높이	$\mathcal{L}(\theta)$
가장 가파른 방향	$-\nabla\mathcal{L}(\theta)$
한 걸음 크기	η (학습률)
새 위치	$\theta^{(t+1)}$

경사하강법의 알고리즘

업데이트 규칙:

$$\theta^{(t+1)} = \theta^{(t)} - \eta\nabla\mathcal{L}(\theta^{(t)})$$

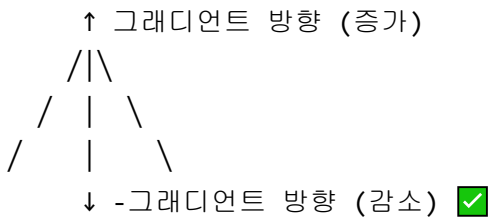
> 💡 **이 식을 한국어로 번역하면:**
>
> **다음 파라미터 = 현재 파라미터 - (학습률 × 그래디언트)**
>
> **각 기호 의미:**
>
> - $\theta^{(t)}$: t번째 스텝의 파라미터
> - 위 첨자 (t)는 시간/반복 횟수를 의미 (제곱 아님!)
> - 예: $\theta^{(0)}$ = 초기값, $\theta^{(1)}$ = 1번 업데이트 후
> - η (에타): **학습률 (learning rate)**
> - "한 걸음의 크기"
> - 보통 0.001, 0.01, 0.1 같은 작은 양수
> - $\nabla\mathcal{L}(\theta^{(t)})$: 현재 위치에서의 그래디언트
> - "어느 방향으로 가면 손실이 증가하는지"

> **구체적 예시:**
>
> 현재 파라미터: $\theta^{(t)} = 5$
> 그래디언트: $\nabla L(\theta^{(t)}) = 2$
> 학습률: $\eta = 0.1$
>
> 업데이트:
> $\theta^{(t+1)} = 5 - 0.1 \times 2 = 5 - 0.2 = 4.8$
>
> → 5에서 4.8로 이동!

왜 마이너스(-)인가?

그래디언트는 **증가하는 방향**을 가리킵니다.
우리는 **감소**시키고 싶으므로 **반대 방향**으로 갑니다.

> 💡 **쉬운 이해:**
>
> 그래디언트 = +2 (오른쪽으로 가면 증가)
> 우리의 이동 = -2 (왼쪽으로 가서 감소) ✅
>
> 그래디언트 = -3 (왼쪽으로 가면 증가)
> 우리의 이동 = +3 (오른쪽으로 가서 감소) ✅



전체 알고리즘:

입력: 손실함수 L, 학습률 η, 초기 파라미터 $\theta^{(0)}$
출력: 최적 파라미터 θ^*

1. t = 0으로 초기화
2. 반복:
   a. 그래디언트 계산: $g^{(t)} = \nabla L(\theta^{(t)})$
   b. 파라미터 업데이트: $\theta^{(t+1)} = \theta^{(t)} - \eta \cdot g^{(t)}$
   c. t = t + 1
3. 종료 조건 만족시 $\theta^{(t)}$ 반환

종료 조건:

- 그래디언트가 충분히 작음: $||\nabla \mathcal{L}(\theta)|| < \epsilon$
- 손실 변화가 작음: $|\mathcal{L}^{(t)} - \mathcal{L}^{(t-1)}| < \epsilon$
- 최대 반복 횟수 도달

구체적 예시: 1차원 함수

문제:

$$\min_{\theta} \mathcal{L}(\theta) = (\theta - 3)^2$$

그래디언트:

$$\frac{d\mathcal{L}}{d\theta} = 2(\theta - 3)$$

초기값: $\theta^{(0)} = 0$

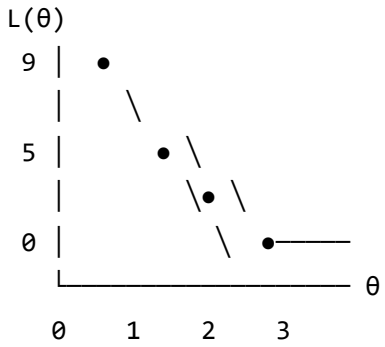
학습률: $\eta = 0.1$

반복:

t	$\theta^{(t)}$	$\mathcal{L}(\theta^{(t)})$	$\nabla \mathcal{L}$	업데이트
0	0	9	-6	$0 - 0.1 \cdot (-6) = 0.6$
1	0.6	5.76	-4.8	$0.6 - 0.1 \cdot (-4.8) = 1.08$
2	1.08	3.69	-3.84	$1.08 - 0.1 \cdot (-3.84) = 1.46$
3	1.46	2.36	-3.07	...
...	...	...	...	...

t	$\theta^{(t)}$	$\mathcal{L}(\theta^{(t)})$	$\nabla\mathcal{L}$	**업데이트**
∞	3	0	0	수렴 ✅

시각화:



● = 각 반복의 위치
점점 최소값 θ=3으로 수렴

다차원 경사하강법

2차원 예시:

$$\min_{w,b} \mathcal{L}(w,b) = (w-2)^2 + (b-1)^2$$

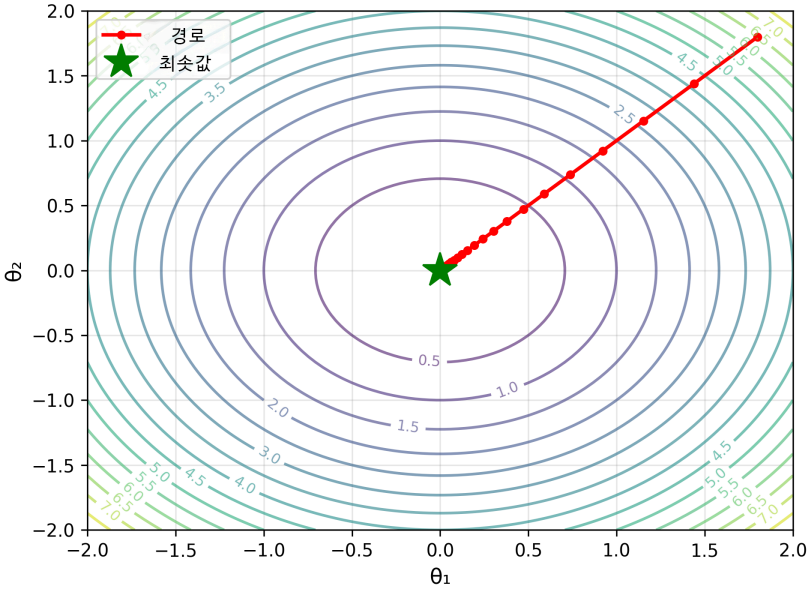
그래디언트:

$$\nabla\mathcal{L} = \begin{bmatrix} \frac{\partial\mathcal{L}}{\partial w} \\ \frac{\partial\mathcal{L}}{\partial b} \end{bmatrix} = \begin{bmatrix} 2(w-2) \\ 2(b-1) \end{bmatrix}$$

업데이트:

$$\begin{bmatrix} w^{(t+1)} \\ b^{(t+1)} \end{bmatrix} = \begin{bmatrix} w^{(t)} \\ b^{(t)} \end{bmatrix} - \eta \begin{bmatrix} 2(w^{(t)}-2) \\ 2(b^{(t)}-1) \end{bmatrix}$$

등고선 시각화:



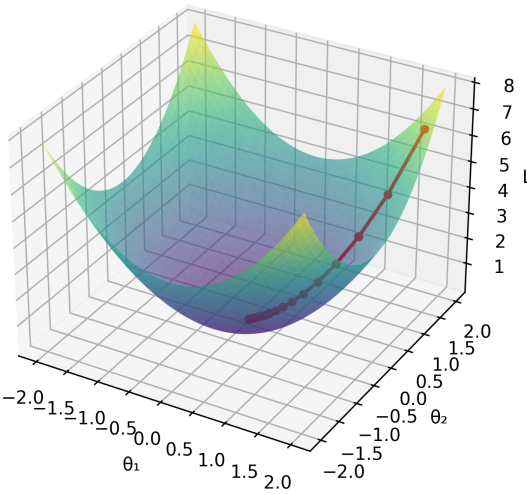


그림 설명:

- **왼쪽**: 2D 등고선 그래프에서 경사하강법의 경로. 빨간 점들이 각 반복의 위치이며, 녹색 별이 최소값입니다.
- **오른쪽**: 3D 손실 함수 표면. 경사하강법이 산을 내려가듯이 최소값으로 수렴하는 것을 보여줍니다.

학습률 (Learning Rate)의 중요성

학습률 η는 경사하강법에서 **가장 중요한 하이퍼파라미터**입니다.

학습률이 너무 작으면 ⏳

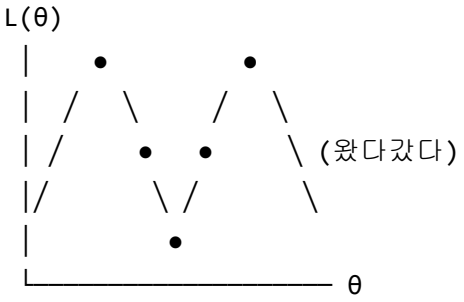
$$\eta \text{ too small} \rightarrow \text{수렴이 매우 느림}$$

```
L(θ)
 │  •
 │   • (한 걸음이 너무 작음)
 │    •
 │     •
 │      • ... (천천히 이동)
 │       •
 │        •—
 └──────────── θ
```

수천 번 반복해도 최소값에 못 도달 😴

학습률이 너무 크면 💥

$$\eta \text{ too large} \rightarrow \text{발산하거나 진동}$$



최소값을 지나쳐서 튕겨나감! 💥
발산하거나 영원히 진동

적절한 학습률 ✅

$$\eta \text{ just right} \rightarrow \text{빠르고 안정적 수렴}$$

시각화: 학습률에 따른 수렴 속도 비교

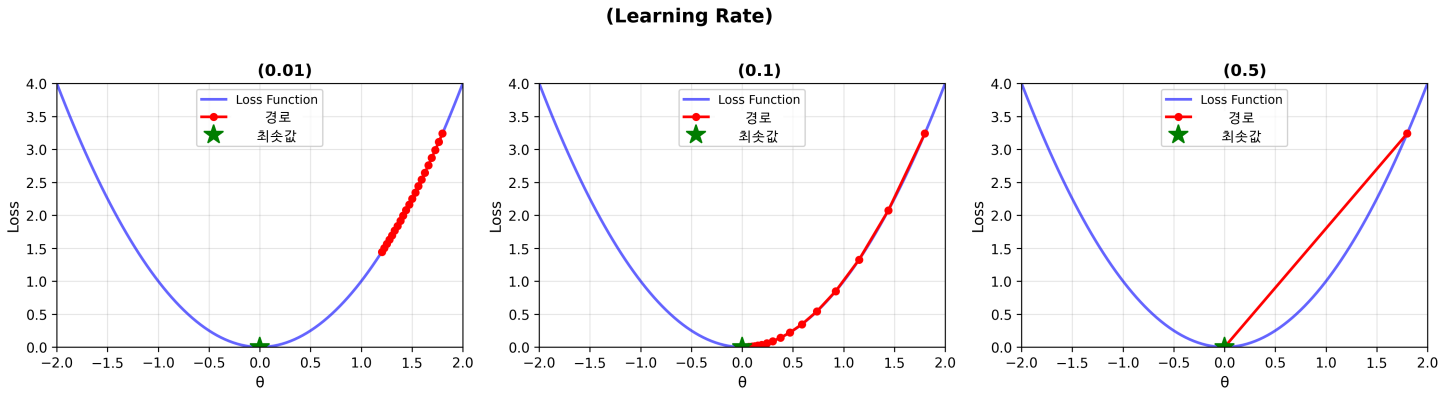


그림 설명:

- **왼쪽 (0.01)**: 학습률이 너무 작아서 매우 느리게 수렴. 20번 반복해도 최소값에 도달하지 못함.
- **중간 (0.1)**: 적절한 학습률로 빠르고 안정적으로 수렴. 이상적!
- **오른쪽 (0.5)**: 학습률이 너무 커서 최소값을 계속 지나치며 발산. 수렴하지 못함.

학습률 선택 전략

1. Grid Search

$$\eta \in \{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$$

각각 시도해보고 가장 좋은 것 선택.

2. Learning Rate Decay (학습률 감소)

반복이 진행될수록 학습률을 줄임:

$$\eta^{(t)} = \frac{\eta_0}{1 + \alpha t}$$

또는 지수적 감소:

$$\eta^{(t)} = \eta_0 \cdot \beta^t \quad (\beta < 1)$$

직관:

- 초기: 큰 걸음으로 빠르게 이동
- 후기: 작은 걸음으로 정밀하게 조정

3. Adaptive Learning Rate

각 파라미터마다 다른 학습률 사용 (Adam, RMSProp 등)

경사하강법의 변형들

기본 경사하강법에는 여러 문제가 있습니다. 이를 개선한 변형들을 살펴봅시다.

1️⃣ 배치 경사하강법 (Batch Gradient Descent)

지금까지 설명한 기본 방법

전체 데이터셋을 사용하여 그래디언트 계산:

$$\nabla \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla L(y_i, f_\theta(x_i))$$

장점:

- ✅ 정확한 그래디언트
- ✅ 안정적인 수렴

단점:

- ❌ 데이터가 많으면 **매우 느림**
- ❌ 메모리 부족 가능

예시:

데이터 100만 개면, 한 번의 업데이트에 100만 개 모두 처리해야 함!

2 확률적 경사하강법 (Stochastic Gradient Descent, SGD)

하나의 데이터만 사용:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(y_i, f_{\theta^{(t)}}(x_i))$$

알고리즘:

```
For each epoch:
    데이터를 섞는다 (shuffle)
    For each 데이터 (xᵢ, yᵢ):
        그래디언트 계산: g = ▽L(yᵢ, f(xᵢ))
        업데이트: θ = θ - η·g
```

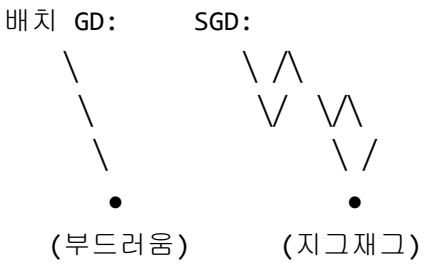
장점:

- ✅ **매우 빠름** (데이터 1개씩만 처리)
- ✅ 메모리 효율적
- ✅ 지역 최소값에서 탈출 가능 (노이즈 때문에)

단점:

- ❌ 그래디언트가 **noisy** (진동)
- ❌ 수렴이 불안정

시각화:



3 미니배치 경사하강법 (Mini-batch Gradient Descent)

배치 GD와 SGD의 절충안

작은 배치(b개)의 데이터 사용:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla L(y_i, f_{\theta^{(t)}}(x_i))$$

알고리즘:

```
For each epoch:
    데이터를 섞는다
    For each mini-batch B (크기 b):
        그래디언트 계산: g = (1/b) Σ ▽L(yᵢ, f(xᵢ))
        업데이트: θ = θ - η·g
```

배치 크기: 보통 $b \in \{32, 64, 128, 256, 512\}$

장점:

- ✅ 배치 GD보다 **빠름**
- ✅ SGD보다 **안정적**
- ✅ GPU 병렬화 효율적

단점:

- ⚠️ 배치 크기 선택 필요

실전에서의 표준

현대 딥러닝에서는 **미니배치가 기본**입니다!

비교 표

방법	배치 크기	속도	안정성	메모리
Batch GD	n (전체)	느림	높음	많음
SGD	1	빠름	낮음	적음
Mini-batch	b (보통 32-256)	중간	중간	중간

4 모멘텀 (Momentum)

문제: SGD는 진동이 심함

해결: "관성"을 추가

업데이트 규칙:

$$v^{(t+1)} = \gamma v^{(t)} + \eta \nabla \mathcal{L}(\theta^{(t)}) \tag{44}$$
$$\theta^{(t+1)} = \theta^{(t)} - v^{(t+1)} \tag{45}$$

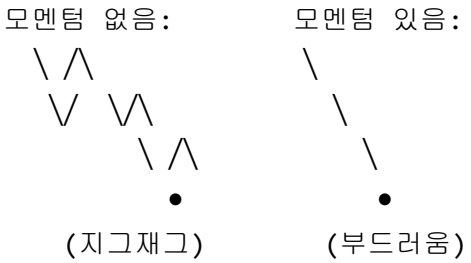
여기서:

- v: velocity (속도)
- γ: 모멘텀 계수 (보통 0.9)

직관:

공이 언덕을 굴러 내려갈 때, 이전 속도를 기억합니다.

- 같은 방향: 가속
- 반대 방향: 감속



수식 풀이:

$v^{(t)}$를 재귀적으로 풀면:

> 💡 **재귀 전개 단계별 과정**
>
> **시작: 기본 식**
>
> $$v^{(t)} = \gamma v^{(t-1)} + \eta \nabla \mathcal{L}(\theta^{(t-1)})$$
>
> **1단계: $v^{(t-1)}$ 대입**
>
> $$v^{(t)} = \gamma[\gamma v^{(t-2)} + \eta \nabla \mathcal{L}(\theta^{(t-2)})] + \eta \nabla \mathcal{L}(\theta^{(t-1)}) \tag{46}$$
> $$= \gamma^2 v^{(t-2)} + \gamma\eta \nabla \mathcal{L}(\theta^{(t-2)}) + \eta \nabla \mathcal{L}(\theta^{(t-1)}) \tag{47}$$
>
> **2단계: $v^{(t-2)}$ 대입**
>
> $$v^{(t)} = \gamma^2[\gamma v^{(t-3)} + \eta \nabla \mathcal{L}(\theta^{(t-3)})] + \gamma\eta \nabla \mathcal{L}(\theta^{(t-2)}) + \eta \nabla \mathcal{L}(\theta^{(t-1)}) \tag{48}$$
> $$= \gamma^3 v^{(t-3)} + \gamma^2\eta \nabla \mathcal{L}(\theta^{(t-3)}) + \gamma\eta \nabla \mathcal{L}(\theta^{(t-2)}) + \eta \nabla \mathcal{L}(\theta^{(t-1)}) \tag{49}$$

> **패턴 발견:**
>
> 계속 전개하면 ($v^{(0)} = 0$ 가정):
>
> $$v^{(t)} = \eta\nabla\mathcal{L}(\theta^{(t-1)}) + \gamma\eta\nabla\mathcal{L}(\theta^{(t-2)}) + \gamma^2\eta\nabla\mathcal{L}(\theta^{(t-3)}) + \gamma^3\eta\nabla\mathcal{L}(\theta^{(t-4)}) + \ldots$$
>
> **왜 "지수 가중 평균"인가?**
>
> 각 과거 그래디언트에 붙는 계수:
>
> - 1단계 전: η (계수 = 1)
> - 2단계 전: $\gamma\eta$ (계수 = γ^1 = 0.9)
> - 3단계 전: $\gamma^2\eta$ (계수 = γ^2 = 0.81)
> - 4단계 전: $\gamma^3\eta$ (계수 = γ^3 = 0.729)
>
> **구체적 예시 ($\gamma = 0.9$):**
>
> | 시간 | 과거로부터 거리 | 가중치 | |
> |---|---|---|---|
> | t-1 | 0 | 1.0 | (100%) |
> | t-2 | 1 | 0.9 | (90%) |
> | t-3 | 2 | 0.81 | (81%) |
> | t-4 | 3 | 0.729 | (73%) |
> | t-5 | 4 | 0.656 | (66%) |
> | t-10 | 9 | 0.387 | (39%) |
>
> 과거로 갈수록 **지수적으로 감소** → "지수 가중 평균"
>
> **직관:**
> 최근 방향은 중요하게 (100%), 오래된 방향은 점점 잊어감 (39%...)

과거 그래디언트의 **지수 가중 평균**!

5 Adam (Adaptive Moment Estimation)

현대 딥러닝의 표준 옵티마이저

왜 Adam이 필요한가?

모멘텀의 한계:

- 모든 파라미터에 **같은 학습률** 적용
- 문제: 어떤 파라미터는 빠르게, 어떤 파라미터는 천천히 학습해야 함!

생활 속 비유:

```
상황: 두 사람이 산에 오르는데...

파라미터 A (체력 좋음):
  - 가파른 길도 큰 보폭으로 가능
  - 학습률 높여도 OK

파라미터 B (체력 약함):
  - 가파른 길에서 비틀거림
  - 학습률 낮춰야 안전

→ 각자에게 맞는 보폭이 필요!
```

Adam의 해결책:

모멘텀 + **적응적 학습률** (파라미터마다 다르게!)

업데이트 규칙:

$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1)\nabla\mathcal{L}(\theta^{(t)}) \quad (1\text{차 모멘트}) \tag{50}$$

$$v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2)(\nabla\mathcal{L}(\theta^{(t)}))^2 \quad (2\text{차 모멘트}) \tag{51}$$

$$\hat{m}^{(t)} = \frac{m^{(t)}}{1-\beta_1^t} \quad (\text{편향 보정}) \tag{52}$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1-\beta_2^t} \quad (\text{편향 보정}) \tag{53}$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta\frac{\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}}+\epsilon} \tag{54}$$

> 💡 **각 항목이 하는 일**
>
> **1. $m^{(t)}$ (1차 모멘트 = 평균 방향)**
>
> "어느 방향으로 가야 하는가?"
>
> - 그래디언트의 **지수 가중 평균**
> - 모멘텀과 같은 역할
> - $\beta_1 = 0.9$: 과거 90% + 현재 10%
>
> **2. $v^{(t)}$ (2차 모멘트 = 변동성)**
>
> "이 파라미터가 얼마나 요동치는가?"
>
> - 그래디언트 **제곱**의 지수 가중 평균

- 변동이 크면 → $v^{(t)}$ 큼
- 변동이 작으면 → $v^{(t)}$ 작음
- $\beta_2 = 0.999$: 과거 99.9% + 현재 0.1% (더 천천히!)

구체적 예시:

```
파라미터 A의 그래디언트 히스토리:
  [5.0, 5.1, 4.9, 5.0, 5.2]  → 안정적!
  v^(t) = 25 정도 (작음)

파라미터 B의 그래디언트 히스토리:
  [10, -8, 12, -9, 11]  → 요동침!
  v^(t) = 100 정도 (큼)
```

3. $\hat{m}^{(t)}, \hat{v}^{(t)}$ (편향 보정)

"왜 필요한가?"

문제: $m^{(0)} = 0, v^{(0)} = 0$으로 초기화
→ 초기에는 $m^{(t)}, v^{(t)}$가 0에 치우침 (편향됨)

구체적 계산:

```
초기 단계 (t=1):
  m^(1) = 0.9 × 0 + 0.1 × g^(1) = 0.1 g^(1)
  → 실제 그래디언트의 10%만 반영! (너무 작음)

편향 보정 후:
  m̂^(1) = 0.1 g^(1) / (1 - 0.9^1) = 0.1 g^(1) / 0.1 = g^(1)
  → 원래 크기로 복원! ✅
```

시간이 지나면 (t 커지면):

- $\beta_1^t \to 0$ (0.9^100 ≈ 0.00003)
- $1 - \beta_1^t \to 1$
- 편향 보정 효과 사라짐 (더 이상 필요 없음)

4. 최종 업데이트

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}} + \epsilon}$$

- 분자 $\hat{m}^{(t)}$: "어느 방향으로 갈까"
- 분모 $\sqrt{\hat{v}^{(t)}}$: "얼마나 조심스럽게 갈까"
  - 변동 크면 → 분모 커짐 → 보폭 작아짐 (조심!)
  - 변동 작으면 → 분모 작아짐 → 보폭 커짐 (대담하게!)
- $\epsilon = 10^{-8}$: 0으로 나누기 방지

구체적 예시:

```
파라미터 A (안정적):
  m̂ = 5.0, v̂ = 25
  업데이트 = 0.001 × 5.0 / √25 = 0.001 × 1.0 = 0.001

파라미터 B (요동침):
  m̂ = 5.0, v̂ = 100
  업데이트 = 0.001 × 5.0 / √100 = 0.001 × 0.5 = 0.0005
  → 절반만 움직임! (더 조심스럽게)
```

기본 하이퍼파라미터:

- $\beta_1 = 0.9$: 1차 모멘트 감쇠 (방향의 관성)
- $\beta_2 = 0.999$: 2차 모멘트 감쇠 (변동성의 관성, 더 느리게)
- $\epsilon = 10^{-8}$: 수치 안정성
- $\eta = 0.001$: 기본 학습률

장점:

- ✅ 각 파라미터마다 **적응적 학습률**
- ✅ 대부분의 문제에서 **잘 작동**
- ✅ 하이퍼파라미터 튜닝 덜 필요
- ✅ 초기 편향 자동 보정

직관 요약:

```
SGD:        모두 같은 보폭으로
Momentum:   관성을 더해서
Adam:       각자 맞는 보폭으로 + 관성

자주 업데이트되는 파라미터 → 학습률 자동 감소
드물게 업데이트되는 파라미터 → 학습률 자동 증가
요동치는 파라미터 → 학습률 자동 감소 (조심!)
안정적 파라미터 → 학습률 유지 (대담하게!)
```

실전 예시: 선형 회귀

문제:

$$y = 2x + 1 + \epsilon$$

데이터:

$$\{(1, 3.1), (2, 4.9), (3, 7.2), (4, 8.8), (5, 11.1)\}$$

모델:

$$\hat{y} = wx + b$$

손실함수 (MSE):

$$\mathcal{L}(w, b) = \frac{1}{5}\sum_{i=1}^{5}(y_i - (wx_i + b))^2$$

그래디언트:

$$\frac{\partial \mathcal{L}}{\partial w} = -\frac{2}{5}\sum_{i=1}^{5} x_i(y_i - (wx_i + b)) \tag{55}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\frac{2}{5}\sum_{i=1}^{5}(y_i - (wx_i + b)) \tag{56}$$

초기값: $w^{(0)} = 0, b^{(0)} = 0$

학습률: $\eta = 0.01$

반복:

Epoch	w	b	$\mathcal{L}$
0	0.00	0.00	129.86
1	0.60	0.54	25.91
2	1.04	0.90	5.88
5	1.68	1.26	0.62
10	1.92	1.08	0.14

Epoch	w	b	$\mathcal{L}$
20	1.99	1.01	0.09
50	2.00	1.00	0.09

수렴! $w \approx 2, b \approx 1$ ✅

2장 요약 정리 📝

핵심 개념

1. 손실함수 (Loss Function)

```
정의: 모델의 예측이 얼마나 틀렸는지 수치화
역할: 학습의 목표 제공
종류: MSE, MAE, Cross-Entropy, Hinge Loss
```

2. 최적화 (Optimization)

```
정의: 손실을 최소화하는 파라미터 찾기
필요조건: 그래디언트 = 0
볼록성: 지역 최소값 = 전역 최소값 (볼록 함수에서)
```

3. 경사하강법 (Gradient Descent)

```
원리: 그래디언트 반대 방향으로 이동
업데이트: θ(t+1) = θ(t) - η▽L(θ(t))
변형: Batch, SGD, Mini-batch, Momentum, Adam
```

손실함수 선택 가이드

문제	손실함수	이유
회귀	MSE	미분 가능, 볼록
회귀 (이상치)	MAE, Huber	강건성

문제	손실함수	이유
이진 분류	Cross-Entropy	확률 해석
다중 분류	Cross-Entropy	표준

옵티마이저 선택 가이드

옵티마이저	장점	단점	추천 상황
Batch GD	안정적	느림	작은 데이터
SGD	빠름	불안정	온라인 학습
Mini-batch	균형	-	대부분의 경우
Momentum	가속	하이퍼파라미터	SGD 개선
Adam	적응적	메모리	**기본 선택** ✅

핵심 질문과 답

Q1: 손실함수가 0이면 완벽한 모델인가?

```
A: 학습 데이터에서는 완벽. 하지만:
   - 과적합 가능성 (다음 챕터)
   - 테스트 데이터에서도 0인지 확인 필요
```

Q2: 학습률은 어떻게 선택하나?

```
A: 1) Grid Search: {0.001, 0.01, 0.1, 1.0} 시도
   2) Learning Rate Finder
   3) 적응적 옵티마이저 (Adam) 사용
```

Q3: 경사하강법이 항상 최적해를 찾나?

```
A: 볼록 함수: Yes ✅
   비볼록 함수 (신경망): 지역 최소값 가능 ⚠️
```

Q4: SGD와 Mini-batch 차이는?

```
A: 배치 크기:
   - SGD: 1개
   - Mini-batch: 32~256개
   Mini-batch가 실전에서 더 나음
```

실전 체크리스트 ✅

학습 시작 전:

- ☐ 손실함수 선택 (문제 유형에 맞게)
- ☐ 옵티마이저 선택 (보통 Adam)
- ☐ 학습률 설정 (보통 0.001)
- ☐ 배치 크기 설정 (32, 64, 128 등)

학습 중:

- ☐ 손실이 감소하는지 확인
- ☐ 너무 느리면: 학습률 증가
- ☐ 발산하면: 학습률 감소
- ☐ 진동하면: Momentum 또는 Adam 사용

학습 후:

- ☐ 수렴했는지 확인 (손실 변화 < ε)
- ☐ 테스트 데이터로 검증

다음 챕터 예고 🔜

Chapter 3에서는:

- **학습의 품질** 이해하기
- 훈련 오차 vs 일반화 오차
- 일반화 갭
- 과적합과 과소적합 (기본 개념)

연결:

```
Chapter 2: 어떻게 학습하는가? (메커니즘)
         ↓
Chapter 3: 학습이 얼마나 좋은가? (품질 평가)
```

🎓 복습 문제

1. MSE와 MAE의 차이는? 언제 각각 사용하나?
2. 그래디언트가 0이면 반드시 최소값인가?
3. 학습률이 너무 크면 어떤 문제가 생기나?
4. Adam이 SGD보다 나은 이유는?
5. 배치 경사하강법과 확률적 경사하강법의 트레이드오프는?

Chapter 3: 학습의 품질 이해하기 🎯

들어가며

Chapter 2에서 우리는 **어떻게** 학습하는지 (손실함수, 최적화, 경사하강법)를 배웠습니다. 이제 **학습이 얼마나 잘 되었는지**를 평가하는 방법을 배울 차례입니다.

머신러닝의 궁극적 질문:

> 🤔 "학습 데이터에서 손실이 0이면 완벽한 모델인가?"
>
> 🤔 "처음 보는 데이터에서도 잘 작동할까?"
>
> 🤔 "모델이 진짜 배운 건가, 아니면 그냥 외운 건가?"

이 질문들에 답하기 위해 세 가지 핵심 개념을 배웁니다:

```
1. 훈련 오차 vs 일반화 오차: 학습과 실전의 차이
2. 일반화 갭: 둘 사이의 간격
3. 과적합과 과소적합: 학습의 실패 사례들
```

3.1 훈련 오차 vs 일반화 오차 📊

두 가지 오차

머신러닝에서는 **두 종류의 오차**를 구분해야 합니다.

훈련 오차 (Training Error)

> 📘 **훈련 오차 (Training Error)**
>
> **학습에 사용한 데이터**에서 측정한 오차

수식:

$$\text{Training Error} = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} L(y_i, f(x_i))$$

> 💡 **쉽게 말하면:**
>
> "학습할 때 본 문제에서 얼마나 틀렸나?"
>
> 학생 비유: 수업 시간에 푼 연습문제 점수

특징:

- ✅ 학습 중에 계속 확인 가능
- ✅ 경사하강법으로 줄일 수 있음
- ⚠️ **낮다고 좋은 모델이 아님!**

일반화 오차 (Generalization Error)

> 📘 **일반화 오차 (Generalization Error)**
>
> **학습에 사용하지 않은 새로운 데이터**에서 측정한 오차

수식:

$$\text{Generalization Error} = \mathbb{E}_{(x,y)\sim P}[L(y, f(x))]$$

> 💡 **기호 설명 (처음 보는 분들을 위해!)**
>
> **1. $\mathbb{E}[\cdot]$ (기댓값, Expectation)**
>
> "평균적으로 어떤 값이 나올까?"
>
> **생활 속 비유: 주사위**

```
주사위 눈금: 1, 2, 3, 4, 5, 6
각각 확률: 1/6씩

기댓값 = 1×(1/6) + 2×(1/6) + ... + 6×(1/6)
       = 3.5

의미: "평균적으로 3.5가 나온다"
      (한 번에 3.5가 나오는 게 아님!)
```

일반 공식:

$$\mathbb{E}[X] = \sum_{\text{모든 가능한 } x} x \times P(x)$$

- 각 값 × 그 값이 나올 확률 → 모두 더함

머신러닝에서:

```
E[(x,y)~P][L(y, f(x))]
= "모든 가능한 (x,y) 데이터에 대해
   손실 L(y, f(x))의 평균"
```

2. $(x, y) \sim P$ (분포에서 추출)

"∼" 기호 = "~에서 뽑힌다" (drawn from)

생활 속 비유: 복권함

```
상황: 복권함 P가 있음
  - 10원권 70장 (70%)
  - 100원권 25장 (25%)
  - 1000원권 5장 (5%)

"(x,y) ~ P" 의미:
  → 이 복권함에서 랜덤하게 하나 뽑기

뽑을 때마다:
  - 10원이 나올 확률: 70%
  - 100원이 나올 확률: 25%
  - 1000원이 나올 확률: 5%
```

머신러닝에서:

"분포 P" = 실세계 데이터 생성 과정

예: 고양이 vs 개 분류
  P = {
    고양이 사진 나올 확률: 60%
    개 사진 나올 확률: 40%
  }

"(x,y) ~ P" = "실세계에서 데이터 하나 랜덤하게 얻기"

3. 전체 수식 이해하기

$$\text{Generalization Error} = \mathbb{E}_{(x,y)\sim P}[L(y, f(x))]$$

단계별 해석:

1. (x,y) ~ P
   → 실세계 분포에서 데이터 하나 뽑음

2. L(y, f(x))
   → 그 데이터에 대한 손실 계산

3. E[...]
   → 모든 가능한 데이터에 대해 평균

결과: "평균적으로 얼마나 틀리는가"

구체적 예시:

```
문제: 동전 앞뒷면 예측

분포 P:
  - 앞면(H) 나올 확률: 60%
  - 뒷면(T) 나올 확률: 40%

모델 f: 항상 "앞면" 예측

손실 L: 틀리면 1, 맞으면 0

Generalization Error:
  = E[(x,y)~P][L(y, f(x))]
  = 0.6 × 0 (앞면일 때: 맞음)
    + 0.4 × 1 (뒷면일 때: 틀림)
  = 0.4

의미: "평균적으로 40% 틀림"
```

왜 "기댓값"을 사용하는가?

문제: 실세계의 모든 데이터를 볼 수 없음!
해결: 확률적으로 "평균 성능" 추정

쉽게 말하면:

"처음 보는 문제에서 평균적으로 얼마나 틀릴까?"

학생 비유: 여러 번 시험 봤을 때 평균 점수

실전에서는:

진짜 분포 P를 모르므로, **테스트 세트**로 근사:

$$\text{Test Error} \approx \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} L(y_i, f(x_i))$$

특징:

- ✅ **진짜 성능** 측정
- ✅ 우리가 정말 관심 있는 것
- ⚠️ 테스트 데이터는 학습에 절대 사용 금지!

생활 속 비유 🎓

상황: 수능 준비

```
학생 A:
- 수업 연습문제: 100점 (훈련 오차 = 0)
- 실제 수능: 60점 (일반화 오차 = 큼)
→ 문제집만 달달 외움 (과적합)

학생 B:
- 수업 연습문제: 85점 (훈련 오차 = 작음)
- 실제 수능: 82점 (일반화 오차 = 작음)
→ 개념을 이해함 (좋은 일반화)

누가 더 나은 학생일까? → 학생 B! ✅
```

핵심:

- 연습문제 점수 (훈련 오차)가 높다고 실력이 좋은 게 아님
- 실제 시험 점수 (일반화 오차)가 중요!

데이터 분할 전략

머신러닝에서는 데이터를 세 부분으로 나눕니다:

```
전체 데이터
    │
    ├─ 훈련 세트 (Training Set): 60-80%
    │   └─ 모델 학습에 사용
    │
    ├─ 검증 세트 (Validation Set): 10-20%
    │   └─ 하이퍼파라미터 튜닝, 모델 선택
    │
    └─ 테스트 세트 (Test Set): 10-20%
        └─ 최종 성능 평가 (한 번만!)
```

각 세트의 역할:

세트	용도	언제 사용	횟수
훈련 세트	파라미터 학습	경사하강법	수천~수만 번

세트	용도	언제 사용	횟수
검증 세트	모델 선택	여러 모델 비교	여러 번
테스트 세트	최종 평가	제출 전	**1번만!**

> ⚠️ **중요: 테스트 세트를 절대 건드리지 마세요!**
>
> 테스트 세트를 학습이나 모델 선택에 사용하면:
>
> - 오차 추정이 낙관적으로 치우침 (실제보다 좋아 보임)
> - 진짜 일반화 성능을 모르게 됨
>
> **원칙:** 테스트 세트는 마지막 순간까지 봉인! 🔒

수식으로 정리

기호 정의:

- $\mathcal{D}_{\text{train}}$: 훈련 데이터
- $\mathcal{D}_{\text{test}}$: 테스트 데이터
- f: 훈련 데이터로 학습한 모델

훈련 오차:

$$\hat{R}_{\text{train}}(f) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} L(y, f(x))$$

테스트 오차 (일반화 오차의 추정):

$$\hat{R}_{\text{test}}(f) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_{\text{test}}} L(y, f(x))$$

진짜 일반화 오차:

$$R(f) = \mathbb{E}_{(x,y)\sim P_{\text{data}}}[L(y, f(x))]$$

관계:

$$\hat{R}_{\text{test}}(f) \approx R(f) \quad \text{(테스트 세트가 충분히 크면)}$$

3.2 일반화 갭 (Generalization Gap) 📏

일반화 갭이란?

> 📘 **일반화 갭 (Generalization Gap)**
>
> 훈련 오차와 일반화 오차(테스트 오차) 사이의 차이

수식:

$$\text{Generalization Gap} = \hat{R}_{\text{test}}(f) - \hat{R}_{\text{train}}(f)$$

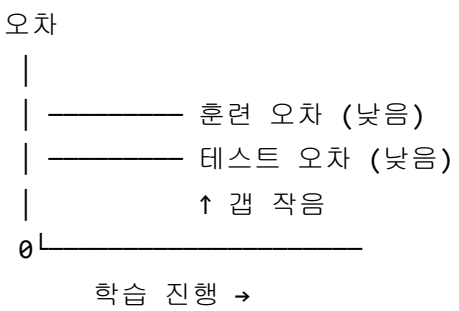
> 💡 **쉽게 말하면:**
>
> "연습문제 점수와 실전 시험 점수의 차이"
>
> - 갭이 작으면: 연습한 대로 실전에서도 잘함 ✅
> - 갭이 크면: 연습문제만 달달 외움 ❌

이상적인 상황

```
훈련 오차:  낮음 ✅
테스트 오차: 낮음 ✅
일반화 갭:  작음 ✅

→ 완벽! 🎉
```

시각화:



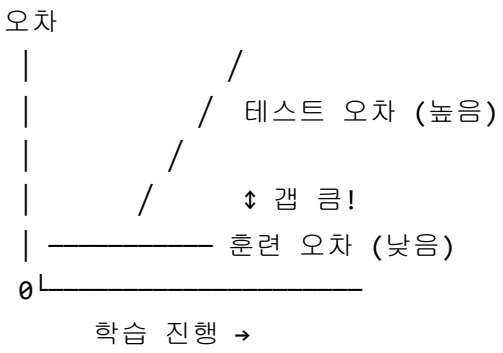
문제 상황들

상황 1: 과적합 (Overfitting)

```
훈련 오차:  매우 낮음 (≈0)
테스트 오차: 높음
일반화 갭:  매우 큼 ❌

→ 암기만 함, 이해 못함
```

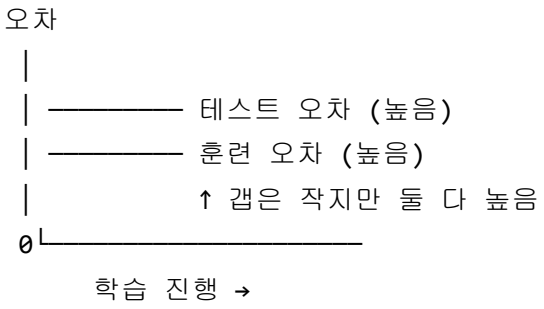
시각화:



상황 2: 과소적합 (Underfitting)

```
훈련 오차:  높음
테스트 오차: 높음
일반화 갭:  작음
→ 둘 다 못함, 학습 자체가 안 됨 ❌
```

시각화:



일반화 갭에 영향을 주는 요인

1. 모델 복잡도 (Model Complexity)

복잡한 모델일수록 일반화 갭이 커지는 경향:

모델 복잡도	훈련 오차	테스트 오차	갭
매우 단순 (직선)	높음	높음	작음
적절 (2차 곡선)	낮음	낮음	작음 ✅
매우 복잡 (100차)	거의 0	높음	큼 ❌

시각화: 모델 복잡도에 따른 훈련/테스트 오차

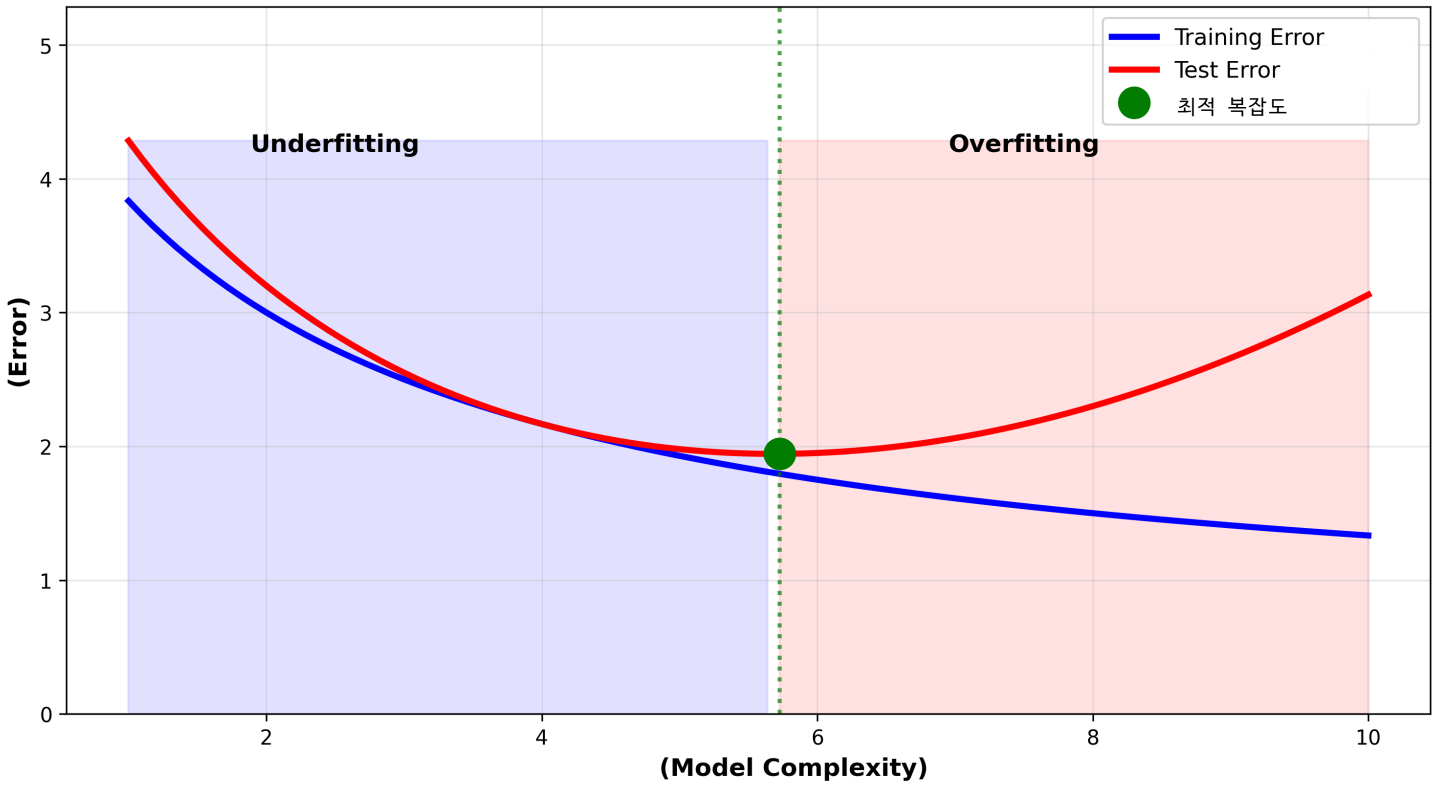


그림 설명: 모델 복잡도가 증가하면서 훈련 오차는 계속 감소하지만, 테스트 오차는 최적점 이후 다시 증가합니다. 녹색 점이 최적 복잡도입니다.

시각화: Underfitting vs Good Fit vs Overfitting

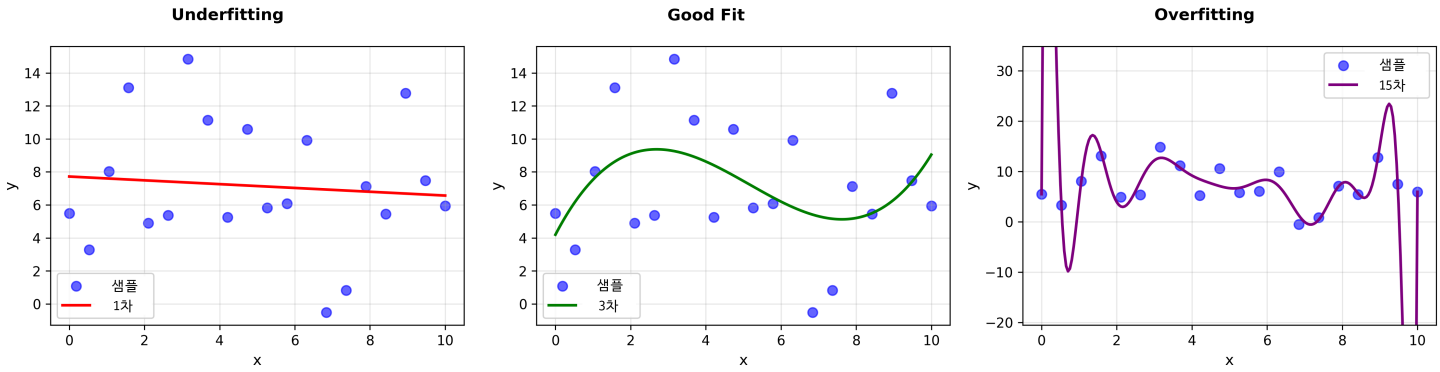


그림 설명:

- **왼쪽 (Underfitting)**: 1차 직선 모델 - 너무 단순하여 데이터 패턴을 제대로 못 잡음
- **중간 (Good Fit)**: 3차 곡선 모델 - 적절한 복잡도로 패턴을 잘 포착
- **오른쪽 (Overfitting)**: 15차 곡선 모델 - 너무 복잡하여 노이즈까지 학습

2. 데이터 크기

데이터가 많을수록 일반화 갭이 작아짐:

$$\text{Gap} \propto \frac{1}{\sqrt{n}}$$

> 💡 **직관:**
>
> 데이터가 많으면:
>
> - 훈련 세트가 실제 분포를 더 잘 대표
> - "외울 것"이 많아서 단순 암기가 어려움
> - 모델이 진짜 패턴을 배워야 함

구체적 예시:

```
데이터 10개:
- 훈련 오차: 0.05
- 테스트 오차: 0.50  (갭 = 0.45, 큼!)

데이터 10,000개:
- 훈련 오차: 0.10
- 테스트 오차: 0.15  (갭 = 0.05, 작음!)
```

3. 정규화 (Regularization)

정규화는 모델의 복잡도를 제한하여 갭을 줄임.

나중에 자세히 다룹니다!

일반화 갭 측정 방법

교차 검증 (Cross-Validation)

데이터를 여러 번 나눠서 평균을 구함:

```
K-Fold Cross-Validation (K=5):

Fold 1: [Test][Train][Train][Train][Train]
Fold 2: [Train][Test][Train][Train][Train]
Fold 3: [Train][Train][Test][Train][Train]
Fold 4: [Train][Train][Train][Test][Train]
Fold 5: [Train][Train][Train][Train][Test]

→ 5개의 테스트 오차 평균
```

장점:

- ✅ 데이터를 효율적으로 사용
- ✅ 더 신뢰할 수 있는 추정

3.3 과적합과 과소적합 🎭

학습의 두 가지 실패

머신러닝 모델이 실패하는 방식은 크게 두 가지입니다.

과소적합 (Underfitting)

> 📘 **과소적합 (Underfitting)**
>
> 모델이 너무 단순해서 데이터의 패턴을 **제대로 학습하지 못함**

특징:

- 훈련 오차: 높음 ❌
- 테스트 오차: 높음 ❌
- 일반화 갭: 작음

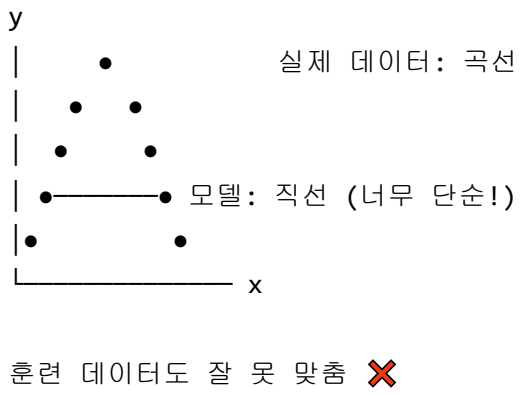
생활 비유:

```
시험 준비:
"공부를 너무 대충 해서 연습문제도 못 풀고, 시험도 못 봄"

요리 비유:
"레시피를 너무 단순화해서 '물+소금=요리' → 맛없음"
```

그래프 예시:

데이터가 2차 곡선인데 직선으로 학습:



훈련 데이터도 잘 못 맞춤 ❌

수학적으로:

$$\text{True function: } f^*(x) = x^2$$

$$\text{Model: } f(x) = wx + b \quad (\text{선형})$$

→ 모델이 2차 함수를 표현할 수 없음!

원인:

- 모델이 너무 단순 (파라미터가 너무 적음)
- 학습이 부족 (반복 횟수 부족)
- 특성(feature)이 부족

해결 방법:

1. 더 복잡한 모델 사용
    - 예: 선형 → 다항식
2. 더 많은 특성 추가

3. 학습 시간 늘리기 (더 많은 epoch)

과적합 (Overfitting)

> 📘 **과적합 (Overfitting)**
>
> 모델이 **훈련 데이터를 너무 정확히** 맞추려다가 **노이즈까지 학습**함

특징:

- 훈련 오차: 매우 낮음 (≈0)
- 테스트 오차: 높음 ❌
- 일반화 갭: 매우 큼 ❌

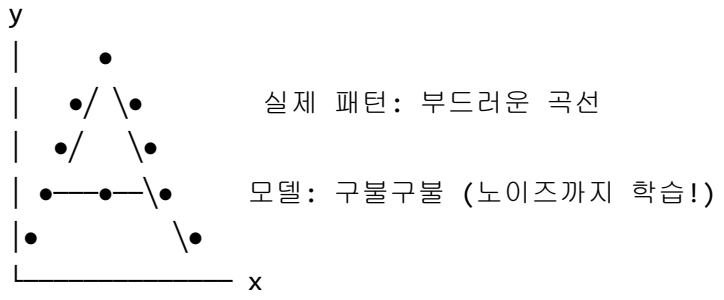
생활 비유:

```
시험 준비:
"연습문제 1000개를 통째로 외워서 연습문제는 100점,
 하지만 실제 시험은 60점"

요리 비유:
"특정 재료(오늘 산 당근)에만 맞춰서 레시피를 만듦
 → 다른 당근으로는 맛이 이상함"
```

그래프 예시:

데이터가 2차 곡선인데 10차 다항식으로 학습:




```
훈련 데이터는 완벽히 맞추지만
새 데이터는 못 맞춤 ❌
```

수학적으로:

데이터: $y = x^2 + \epsilon$ (노이즈 ϵ 포함)

$$\text{Good model: } f(x) = w_2x^2 + w_1x + w_0$$

$$\text{Overfit model: } f(x) = \sum_{i=0}^{100} w_i x^i$$

→ 100차 다항식은 노이즈까지 모두 맞춤!

원인:

- 모델이 너무 복잡 (파라미터가 너무 많음)
- 데이터가 너무 적음
- 학습을 너무 오래 함

해결 방법:

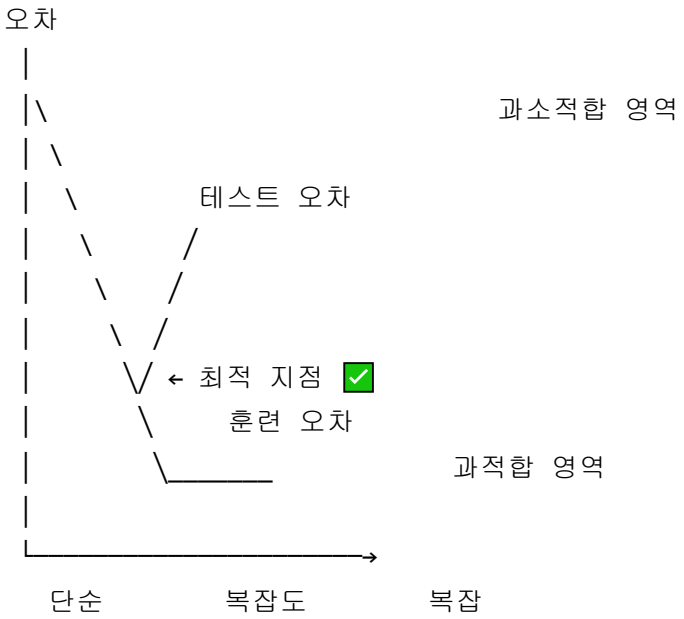
1. **더 많은 데이터** 수집
2. **정규화** (Regularization)
    - L1, L2 정규화
3. **드롭아웃** (Dropout) - 신경망
4. **조기 종료** (Early Stopping)
5. **모델 단순화**
    - 파라미터 수 줄이기

과소적합 vs 과적합 비교표

	과소적합	이상적	과적합
모델 복잡도	너무 낮음	적절	너무 높음
훈련 오차	높음	낮음	매우 낮음 (≈0)
테스트 오차	높음	낮음	높음
일반화 갭	작음	작음	큼
문제	배우질 못함	✅	외우기만 함
해결	복잡도 ↑	-	복잡도 ↓ 또는 데이터 ↑

복잡도-오차 곡선

모델 복잡도에 따른 오차 변화:



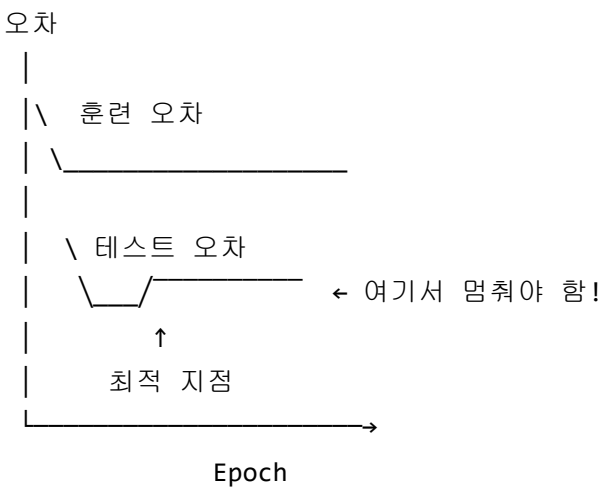
핵심 관찰:

1. **왼쪽 (단순한 모델)**
   - 훈련/테스트 오차 둘 다 높음
   - 과소적합
2. **중간 (적절한 모델)**
   - 훈련/테스트 오차 둘 다 낮음
   - 이상적! ✅
3. **오른쪽 (복잡한 모델)**
   - 훈련 오차: 계속 감소
   - 테스트 오차: 증가!
   - 과적합

과적합 감지 방법

학습 곡선 (Learning Curve) 그리기

에폭(epoch)에 따른 오차 변화:



관찰:

- 초기: 훈련/테스트 오차 둘 다 감소 ✅
- 중간: 테스트 오차가 최소 → **여기서 멈춰야 함!**
- 후기: 훈련 오차는 감소하지만 테스트 오차 증가 → 과적합 ❌

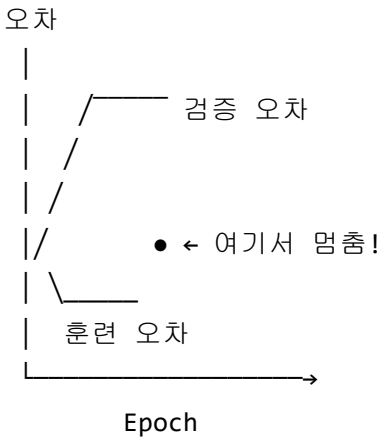
조기 종료 (Early Stopping)

과적합을 방지하는 간단하고 효과적인 방법:

알고리즘:

1. 검증 세트 준비
2. 매 에폭마다:
   a. 훈련 세트로 학습
   b. 검증 세트 오차 측정
   c. 검증 오차가 증가하기 시작하면 → 중지!
3. 검증 오차가 최소였던 모델 선택

시각화:



장점:

- ✅ 구현 간단
- ✅ 효과적
- ✅ 추가 계산 비용 거의 없음

실전 예시: 다항식 회귀

데이터: $y = \sin(2\pi x) + \epsilon$

모델 비교:

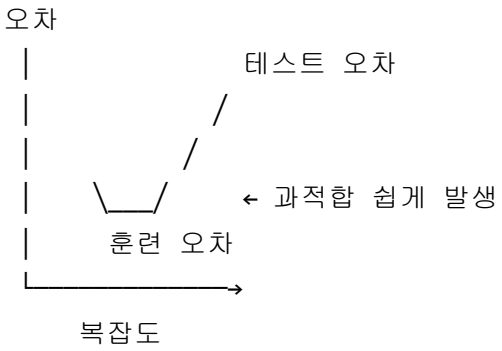
모델	복잡도	훈련 오차	테스트 오차	상태
1차 (선형)	낮음	0.45	0.47	과소적합 ❌
3차 다항식	중간	0.12	0.14	이상적 ✅
9차 다항식	높음	0.01	0.35	과적합 ❌

관찰:

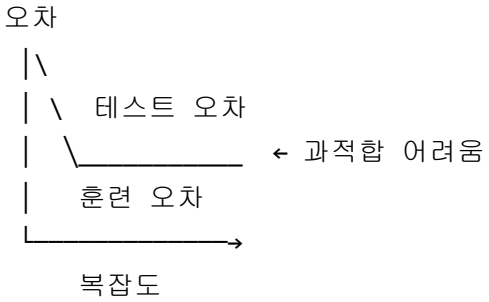
- 1차: 사인 함수를 직선으로 → 못 맞춤
- 3차: 부드러운 곡선으로 → 잘 맞춤
- 9차: 모든 점을 완벽히 맞추지만 → 새 데이터는 못 맞춤

데이터 크기의 영향

적은 데이터 (n=10):



많은 데이터 (n=10,000):



> 💡 **핵심 교훈:**
>
> **데이터는 과적합의 천적이다**
>
> 데이터가 많으면:
>
> - 복잡한 모델 사용 가능
> - 과적합 덜 발생
> - 일반화 성능 향상

3장 요약 정리 📝

핵심 개념

1. 훈련 오차 vs 일반화 오차

```
훈련 오차: 학습 데이터에서의 성능
일반화 오차: 새 데이터에서의 성능 (진짜 실력!)
관계: 일반화 오차 ≥ 훈련 오차 (보통)
```

2. 일반화 갭

```
정의: 테스트 오차 - 훈련 오차
이상적: 갭이 작을수록 좋음
영향 요인: 모델 복잡도, 데이터 크기, 정규화
```

3. 과적합과 과소적합

```
과소적합: 둘 다 못함 (모델이 너무 단순)
과적합: 훈련은 완벽, 테스트는 실패 (모델이 너무 복잡)
목표: 둘 사이의 균형점 찾기
```

데이터 분할 원칙

세트	비율	용도	주의사항
훈련	60-80%	모델 학습	여러 번 봐도 됨
검증	10-20%	모델 선택	여러 모델 비교 가능
테스트	10-20%	최종 평가	**1번만!** 🔒

문제 진단 차트

```
훈련 오차 낮음?
    │
    ├─ No → 과소적합
    │       해결: 모델 복잡도 ↑
    │
    └─ Yes → 테스트 오차 낮음?
             │
             ├─ No → 과적합
             │       해결: 데이터 ↑, 정규화
             │
             └─ Yes → 완벽! ✅
```

핵심 질문과 답

Q1: 훈련 오차가 0이면 좋은 모델인가?

A: 아니요! ❌
- 과적합 가능성 높음
- 테스트 오차를 확인해야 함

Q2: 테스트 세트로 여러 번 평가해도 되나?

A: 안 됩니다! ❌
- 테스트 세트 정보가 모델에 새어 들어감
- 검증 세트로 선택, 테스트 세트는 최종에만

Q3: 과적합을 어떻게 막나?

A: 1) 더 많은 데이터
2) 정규화 (L1, L2)
3) 조기 종료
4) 드롭아웃 (신경망)
5) 모델 단순화

Q4: 과소적합과 과적합 중 뭐가 나은가?

A: 상황에 따라 다르지만:
- 과소적합 → 개선 방법 명확 (복잡도 ↑)
- 과적합 → 데이터 더 필요 (비용 큼)
일반적으로 과소적합이 진단/해결 쉬움

실전 체크리스트 ✅

학습 전:

- ☐ 데이터를 훈련/검증/테스트로 분할
- ☐ 테스트 세트는 따로 보관 (절대 건드리지 말 것!)

학습 중:

- ☐ 훈련 오차와 검증 오차를 함께 모니터링
- ☐ 학습 곡선 그래프 확인
- ☐ 과적합 징후 감지 (검증 오차 증가)

평가:

- [ ] 훈련 오차 < 목표치? → No면 과소적합
- [ ] 검증 오차 < 목표치? → No면 과적합
- [ ] 일반화 갭 < 허용치?
- [ ] **마지막**에 테스트 세트로 최종 평가

다음 챕터 예고 🔜

Chapter 4에서는:

- **모델 성능 평가 방법** 심화
- 다양한 평가 지표 (정확도, 정밀도, 재현율, F1, AUC 등)
- 혼동 행렬 (Confusion Matrix)
- 적절한 지표 선택 방법

연결:

```
Chapter 3: 학습이 얼마나 좋은가? (품질 평가 기초)
         ↓
Chapter 4: 성능을 어떻게 측정하는가? (평가 지표)
```

🎓 복습 문제

1. 훈련 오차와 일반화 오차의 차이는?
2. 일반화 갭이 크다는 것은 무엇을 의미하나?
3. 과적합과 과소적합의 차이를 훈련/테스트 오차로 설명하라.
4. 테스트 세트를 여러 번 사용하면 안 되는 이유는?
5. 과적합을 막는 5가지 방법은?

다음 챕터에서 만나요! 🚀

Chapter 4: 모델의 근본 원리 ⚖️

들어가며

Chapter 3에서 우리는 과적합과 과소적합의 **현상**을 배웠습니다. 이제 **왜 이런 일이 발생하는지**, 모델의 근본적인 원리를 이해할 차례입니다.

이번 챕터의 핵심 질문:

> 🤔 "모델의 '능력'은 무엇으로 결정되나?"
>
> 🤔 "왜 복잡한 모델이 항상 좋은 것은 아닌가?"
>
> 🤔 "일반화 오차는 정확히 어디서 오는가?"

이 질문들에 답하기 위해 세 가지 핵심 개념을 깊이 있게 다룹니다:

- **수용력 (Capacity)**: 모델이 표현할 수 있는 범위
- **편향-분산 트레이드오프**: 일반화 오차의 수학적 분해
- **과적합/과소적합의 이론적 이해**: 왜 발생하고 어떻게 해결하나

4.1 수용력 (Capacity) 🧠

수용력이란?

> 📘 **수용력 (Capacity, Representational Power)**
>
> 모델이 표현할 수 있는 함수의 범위

쉬운 비유: 물통의 크기

```
작은 물통 (낮은 수용력):
└┘  ← 적은 물만 담을 수 있음

큰 물통 (높은 수용력):
│  │
│  │  ← 많은 물을 담을 수 있음
└──┘
```

머신러닝에서:

- **물** = 데이터의 패턴
- **물통** = 모델의 수용력

수학적 정의: 가설 공간 (Hypothesis Space)

왜 가설 공간 개념이 필요한가?

문제: 모델이 아무리 학습해도 **표현할 수 없는** 패턴이 있다!

생활 속 비유: 그림 그리기 도구

```
상황: 원을 그리고 싶음

도구 1: 자 (직선만 가능)
  → 아무리 노력해도 원을 그릴 수 없음!
  → "가설 공간"에 원이 없음

도구 2: 컴퍼스 (원 가능)
  → 원을 그릴 수 있음!
  → "가설 공간"에 원이 있음

핵심: 도구의 한계 = 가설 공간의 한계
```

가설 공간 $\mathcal{H}$:

모델이 **학습을 통해 도달할 수 있는 모든 함수들의 집합**

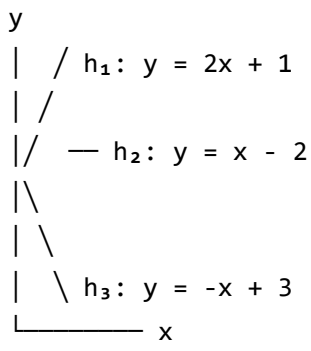
$$\mathcal{H} = \{h_\theta \mid \theta \in \Theta\}$$

> 💡 **수식 기호 설명**
>
> - $\mathcal{H}$ (Calligraphic H): 가설 공간 = "함수들의 집합"
> - h_θ: 파라미터 θ로 만들어지는 하나의 함수
> - Θ (Theta): 가능한 모든 파라미터의 집합
> - $\{... \mid ...\}$: "~인 것들의 집합"
>   - 읽는 법: "θ가 Θ에 속할 때, h_θ들의 집합"
>
> **구체적 예시:**
>
> ```
> 선형 회귀: h(x) = wx + b
>
> Θ = {모든 가능한 (w, b) 쌍}
>   예: (w=2, b=1), (w=-1, b=3), (w=0.5, b=-2), ...
>
> H = {모든 가능한 직선 함수}
>   예: h(x) = 2x+1, h(x) = -x+3, h(x) = 0.5x-2, ...
>
> 의미: "파라미터를 바꾸면서 만들 수 있는 모든 함수"
> ```

예시: 선형 회귀

$$\mathcal{H}_{\text{linear}} = \{h(x) = wx + b \mid w, b \in \mathbb{R}\}$$

이 공간은 **모든 직선**을 포함합니다.

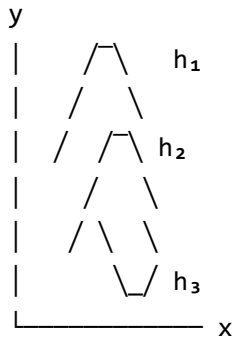


선형 회귀의 가설 공간 = 모든 직선

예시: 다항 회귀 (2차)

$$\mathcal{H}_{\text{poly-2}} = \{h(x) = w_2x^2 + w_1x + w_0 \mid w_0, w_1, w_2 \in \mathbb{R}\}$$

이 공간은 **모든 포물선**을 포함합니다.



다항 회귀의 가설 공간 = 모든 2차 곡선

관계:

$$\mathcal{H}_{\text{linear}} \subset \mathcal{H}_{\text{poly-2}} \subset \mathcal{H}_{\text{poly-3}} \subset \ldots$$

직선은 포물선의 특수한 경우 ($w_2 = 0$)!

수용력에 영향을 주는 요소

1 파라미터 개수

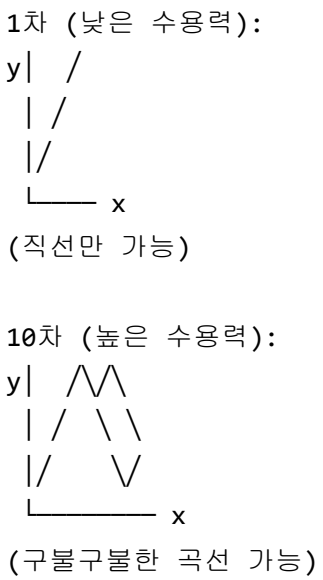
일반적으로:

$$\text{파라미터 많음} \rightarrow \text{수용력 높음}$$

예시: 다항식의 차수

차수	파라미터 수	수용력
1차 (직선)	2개 (w, b)	낮음
2차 (포물선)	3개	중간
10차	11개	높음

시각화:



2 모델 구조

같은 파라미터 수여도 구조에 따라 수용력이 다름!

예시: 신경망 vs 선형 모델

선형 모델 (10개 파라미터):
Input → [w_1...w_{10}] → Output

신경망 (10개 파라미터):
Input → [w_1, w_2] → ReLU → [w_3, w_4] → ... → Output
└──── 비선형 ────┘

→ 신경망의 수용력이 훨씬 높음!
(비선형 변환 덕분)

3 정규화 (Regularization)

정규화는 **수용력을 제한**합니다!

제약 없음:
Capacity ██████████ (높음)

L2 정규화:
Capacity ██████ (제한됨)

강한 L2:
Capacity ███ (매우 제한됨)

수용력의 스펙트럼

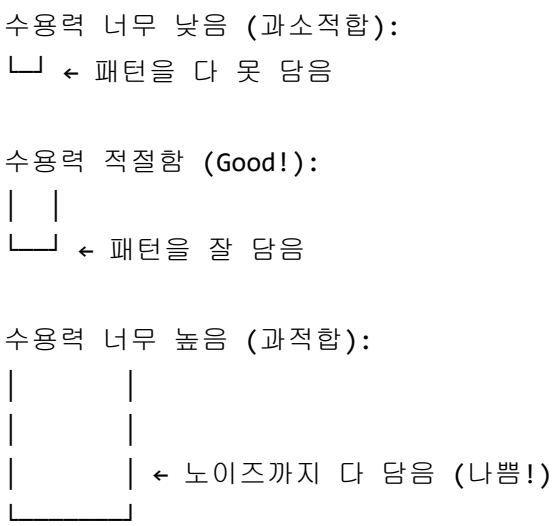
낮음 ←──────────────────────────→ 높음

단순한 모델	복잡한 모델
- 선형 회귀	- 고차 다항식
- 로지스틱 회귀	- 결정 트리 (깊은)
- 나이브 베이즈	- 랜덤 포레스트
	- 신경망 (깊은)
	- k-NN (k=1)
적은 파라미터	많은 파라미터
강한 귀납적 편향	약한 귀납적 편향

최적 수용력 (Optimal Capacity)

문제: 수용력이 높다고 무조건 좋은가?

답: No!



최적 수용력:

$$\text{Optimal Capacity} \approx \text{데이터의 복잡도} + \alpha$$

여기서 α는 약간의 여유 (일반화를 위해)

수용력과 데이터 크기

Universal Approximation Theorem:

> 충분히 큰 신경망은 **어떤** 연속 함수도 근사할 수 있다!

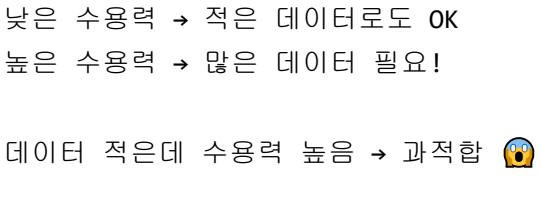
하지만:

이론: 무한 수용력 → 무한 표현력 ✅

현실: 무한 데이터 필요 + 최적화 어려움 ❌

실용적 원칙:

$$\text{필요한 데이터} \propto \text{모델의 수용력}$$



경험 법칙:

파라미터 수	필요한 데이터 (최소)
10개	100 ~ 1,000개
100개	1,000 ~ 10,000개
1,000개	10,000 ~ 100,000개
1,000,000개 (딥러닝)	수백만 ~ 수억 개

> ⚠️ **주의:** 이는 매우 대략적인 가이드입니다!
> 실제로는 문제 복잡도, 노이즈 등에 따라 달라집니다.

4.2 편향-분산 트레이드오프 (Bias-Variance Tradeoff) ⚖️

편향-분산 분해 (Bias-Variance Decomposition)

왜 이 분해가 필요한가?

문제 상황:

```
모델 A: 훈련 오차 = 10%, 테스트 오차 = 15%
모델 B: 훈련 오차 = 1%,  테스트 오차 = 20%

질문: 왜 모델 B가 더 안 좋은가?
      어떻게 고쳐야 하는가?
```

단순히 "오차가 크다"만으로는 문제 진단 불가!

생활 속 비유: 다트 게임

```
목표 ○ = 과녁 중심

선수 A (High Bias, Low Variance):
   ●●
   ●●
        ○
(일관되게 같은 곳에 맞추지만, 과녁에서 벗어남)
→ 조준이 틀렸다! (모델이 너무 단순)

선수 B (Low Bias, High Variance):
●       ●
   ○
      ●
●
(평균적으로는 과녁 근처지만, 매번 다른 곳)
→ 손이 떨린다! (모델이 불안정)

선수 C (High Bias, High Variance):
●
        ●
             ○
  ●
       ●
(조준도 틀리고 손도 떤다)
→ 최악!
```

핵심 통찰:

"실패"에도 **종류**가 있다!

- Bias 문제 → 해결: 더 복잡한 모델
- Variance 문제 → 해결: 더 많은 데이터, 정규화

핵심 질문:
"일반화 오차는 어디서 오는가?"

답: 세 가지 원천!

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

> 💡 **이 식은 머신러닝에서 가장 중요한 식 중 하나입니다!**
>
> **왜?** 오차의 원인을 정확히 진단할 수 있기 때문!

> 💡 **기호 설명**
>
> - y: 실제값 (진짜 정답)
> - $\hat{f}(x)$: 우리 모델의 예측
> - $\mathbb{E}$: 기댓값 (평균) - **여러 학습 데이터셋에 대해**
>
> **"여러 학습 데이터셋"이란?**
>
> 상상 실험:
>
> ```
> 1. 실세계에서 학습 데이터 셋 뽑기 → 모델 학습 → 예측 f₁
> 2. 다시 새로운 학습 데이터 셋 뽑기 → 모델 학습 → 예측 f₂
> 3. 또 다시 새로운 학습 데이터 셋 뽑기 → 모델 학습 → 예측 f₃
> ...
>
> E[f̂(x)] = 이 모든 예측들의 평균
> ```
>
> **실제로는 불가능!** (데이터 하나만 있음)
>
> 하지만 이론적으로 오차의 본질을 이해하는 데 중요!

세 가지 오차 원천

1. Bias (편향)

> 모델의 예측이 평균적으로 정답에서 **얼마나 떨어져 있는가**

$$\text{Bias}[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)] - f(x)$$

여기서:

- $f(x)$: 진짜 함수 (우리가 알고 싶은)
- $\mathbb{E}[\hat{f}(x)]$: 여러 학습 셋으로 학습한 모델들의 평균 예측

직관:

○ = 실제 정답
● = 모델의 예측들

낮은 Bias (Good!):
  ●●
 ●○●
  ●
(정답 주변에 모임)

높은 Bias (Bad!):
○

   ●●●
   ●●
(정답에서 멀리 떨어짐)

Bias가 높은 이유:

- 모델이 **너무 단순**함
- **잘못된 가정** (귀납적 편향이 문제와 안 맞음)
- **과소적합** (Underfitting)

예시:

실제 관계: y = x² (포물선)

선형 모델: y = wx + b

아무리 학습해도 직선밖에 못 만듦
→ 평균적으로 실제 함수와 차이 큼
→ 높은 Bias!

2. Variance (분산)

> 학습 데이터가 바뀌면 모델의 예측이 **얼마나 변하는가**

$$\text{Variance}[\hat{f}(x)] = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$$

직관:

○ = 정답
● = 서로 다른 학습 데이터로 학습한 모델들의 예측

낮은 Variance (Good!):

 ●●
●○●
 ●

(예측들이 모여있음)

높은 Variance (Bad!):

●
    ○   ●
 ●
      ●
   ●

(예측들이 흩어져있음)

Variance가 높은 이유:

- 모델이 **너무 복잡**함
- 학습 데이터의 **노이즈**에 민감
- **과적합** (Overfitting)

예시:

실제 관계: y = x² + 약간의 노이즈

100차 다항식 모델 사용

학습 데이터 A → 구불구불한 곡선 A
학습 데이터 B → 완전히 다른 구불구불한 곡선 B
학습 데이터 C → 또 다른 구불구불한 곡선 C

→ 학습 데이터에 따라 예측이 크게 변함
→ 높은 Variance!

3. Irreducible Error (축소 불가능한 오차)

$$\text{Irreducible Error} = \text{Var}[\epsilon]$$

의미:

- 데이터 자체의 **노이즈**
- 측정 오차
- **어떤 모델로도 줄일 수 없음**

예시:

```
실제 관계: y = x² + ε
                  └ 랜덤 노이즈

아무리 완벽한 모델도
이 노이즈는 예측 못함!
```

편향-분산 트레이드오프

핵심 통찰:

> Bias와 Variance는 **반비례** 관계!

```
Bias를 줄이려면 (복잡한 모델) → Variance 증가
Variance를 줄이려면 (단순한 모델) → Bias 증가
```

시각화:

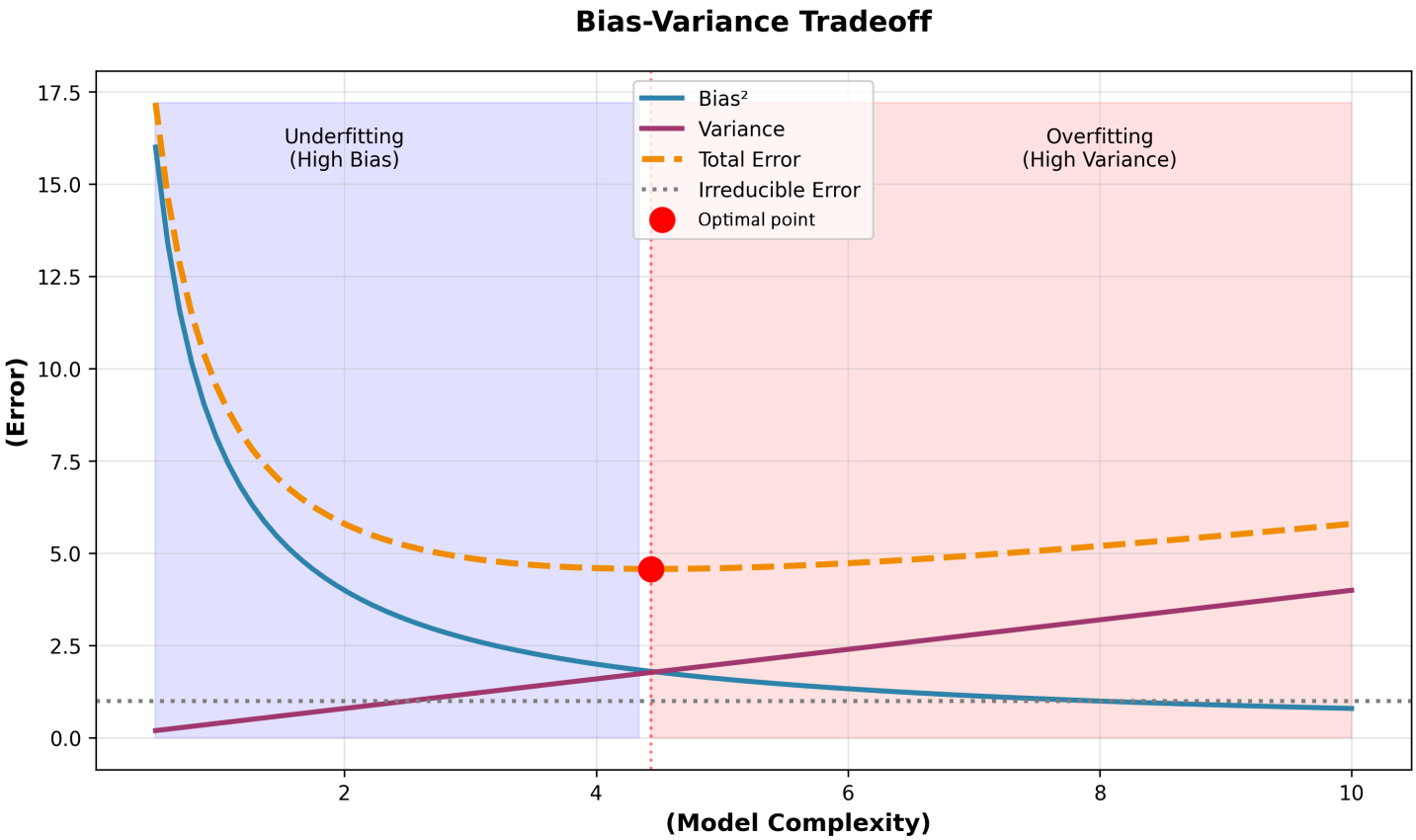


그림 설명: 모델 복잡도에 따른 Bias², Variance, Total Error의 변화. 빨간 점이 최적점(optimal point)으로, Total Error가 최소가 되는 지점입니다.

해석:

단순한 모델 (왼쪽):

- ✅ Variance 낮음 (안정적)
- ❌ Bias 높음 (부정확)
- 결과: **과소적합**

복잡한 모델 (오른쪽):

- ✅ Bias 낮음 (정확)
- ❌ Variance 높음 (불안정)
- 결과: **과적합**

최적 모델 (중간):

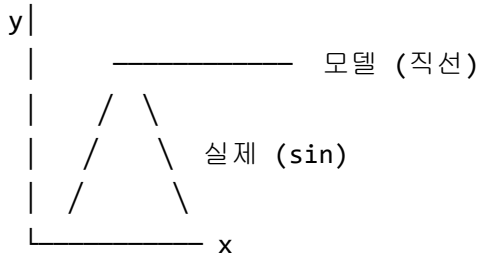
- Bias와 Variance의 **균형**
- **Total Error 최소화** ⭐

구체적 예시로 이해하기

문제: 실제 관계는 $y = \sin(x)$

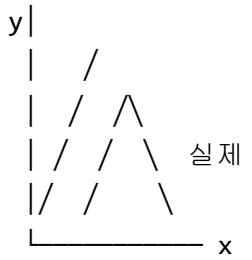
데이터: $x \in [0, 2\pi]$, 노이즈 포함

모델 1: 상수 모델 (y = c)



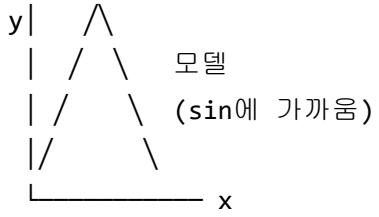
Bias: 매우 높음 (평균적으로 실제와 멀리 떨어짐)
Variance: 0 (항상 같은 예측)

모델 2: 선형 모델 (y = wx + b)



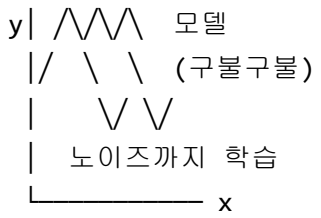
Bias: 높음 (직선은 sin을 표현 못함)
Variance: 낮음 (학습 데이터 바뀌어도 비슷한 직선)

모델 3: 5차 다항식



Bias: 낮음 (sin을 잘 근사)
Variance: 중간 (학습 데이터에 따라 약간 변함)
→ Good! ✅

모델 4: 50차 다항식



Bias: 매우 낮음 (학습 데이터는 완벽히 맞춤)
Variance: 매우 높음 (데이터 바뀌면 완전히 다른 곡선)
→ 과적합! ❌

수학적 유도 (심화) 📐

> 💡 **이 부분은 선택적입니다. 수학적 증명에 관심없다면 건너뛰어도 됩니다.**

목표: $\mathbb{E}[(y - \hat{f}(x))^2]$를 Bias² + Variance로 분해

가정:

- $y = f(x) + \epsilon$, 여기서 $\mathbb{E}[\epsilon] = 0$, $\text{Var}[\epsilon] = \sigma^2$
- $\hat{f}$: 학습 데이터 $\mathcal{D}$로 학습한 모델

유도:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D},\epsilon}[(y - \hat{f}(x))^2] && (57)\\
&= \mathbb{E}[(f(x) + \epsilon - \hat{f}(x))^2] && (58)\\
&= \mathbb{E}[(f(x) - \hat{f}(x))^2 + 2\epsilon(f(x) - \hat{f}(x)) + \epsilon^2] && (59)
\end{aligned}$$

$\mathbb{E}[\epsilon] = 0$이고 ϵ은 $\hat{f}$와 독립이므로:

$$\begin{aligned}
&= \mathbb{E}[(f(x) - \hat{f}(x))^2] + \mathbb{E}[\epsilon^2] && (60)\\
&= \mathbb{E}[(f(x) - \hat{f}(x))^2] + \sigma^2 && (61)
\end{aligned}$$

첫 번째 항을 더 분해:

$\bar{f}(x) = \mathbb{E}[\hat{f}(x)]$라고 하면,

$$\begin{aligned}
&\mathbb{E}[(f(x) - \hat{f}(x))^2] && (62)\\
&= \mathbb{E}[(f(x) - \bar{f}(x) + \bar{f}(x) - \hat{f}(x))^2] && (63)\\
&= \mathbb{E}[(f(x) - \bar{f}(x))^2 + 2(f(x) - \bar{f}(x))(\bar{f}(x) - \hat{f}(x)) + (\bar{f}(x) - \hat{f}(x))^2] && (64)
\end{aligned}$$

중간 항은 0:

$$\mathbb{E}[2(f(x) - \bar{f}(x))(\bar{f}(x) - \hat{f}(x))] = 2(f(x) - \bar{f}(x)) \cdot \mathbb{E}[\bar{f}(x) - \hat{f}(x)] = 0$$

따라서:

$$= (f(x) - \bar{f}(x))^2 + \mathbb{E}[(\bar{f}(x) - \hat{f}(x))^2] \tag{65}$$
$$= \text{Bias}^2[\hat{f}(x)] + \text{Variance}[\hat{f}(x)] \tag{66}$$

최종:

$$\boxed{\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \sigma^2}$$

■

4.3 과적합과 과소적합의 이론적 이해 (심화) 🔬

과소적합 (Underfitting) - 깊이 파기

> 📘 **과소적합**
>
> 모델의 **수용력**이 데이터의 복잡도보다 **낮아서** 패턴을 제대로 학습하지 못함

수학적 관점:

$$\mathcal{H}_{\text{model}} \not\supseteq f^* \quad \text{(진짜 함수가 가설 공간 밖에 있음)}$$

특징:

- 높은 Bias, 낮은 Variance
- 훈련 오차: 높음 ❌
- 테스트 오차: 높음 ❌
- **둘 다 나쁨!**

원인 분석:

1. 수용력 부족

진짜 함수: y = x³
모델: y = wx + b (직선만 표현 가능)

→ 아무리 학습해도 x³을 표현 못함

2. 특징 부족

문제: 집값 예측
사용 특징: 크기만
누락된 특징: 위치, 층수, 연식 등

→ 정보가 부족해서 학습 불가

3. 과도한 정규화

L2 정규화 계수 λ = 100 (너무 큼)

→ 가중치가 거의 0으로 수렴
→ 모델이 거의 상수 함수

해결 방법:

방법	설명	예시
복잡도 증가	더 복잡한 모델	선형 → 다항식
특징 추가	Feature engineering	새로운 변수 생성
정규화 감소	λ 줄이기	λ: 100 → 0.01
더 오래 학습	Epoch 증가	10 → 100 epochs

과적합 (Overfitting) - 깊이 파기

> 📘 **과적합**
>
> 모델의 **수용력**이 데이터의 복잡도보다 **높아서** 노이즈까지 학습함

수학적 관점:

$$\mathcal{H}_{\text{model}} \gg \{f^*\} \quad \text{(가설 공간이 필요 이상으로 큼)}$$

특징:

- 낮은 Bias, 높은 Variance
- 훈련 오차: 매우 낮음 ✅
- 테스트 오차: 높음 ❌
- **일반화 갭이 큼!**

원인 분석:

1. 과도한 수용력

```
데이터: 10개 점
모델: 100차 다항식

→ 파라미터(101개) >> 데이터(10개)
→ 무한히 많은 해가 존재
→ 노이즈까지 맞추는 해 선택
```

2. 데이터 부족

```
파라미터: 10,000개
데이터: 100개

→ 100개 데이터만으로 10,000개 파라미터 결정 불가
→ 과적합 불가피
```

3. 학습 시간 과다

```
Epoch
 0   ← 과소적합
 50  ← 적절
 500 ← 과적합 (훈련 데이터 암기 시작)
```

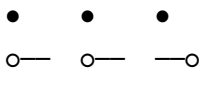
VC 차원 (Vapnik-Chervonenkis Dimension)

> 모델이 **완벽히 분류할 수 있는** 최대 데이터 개수

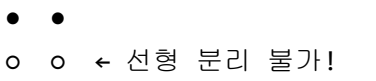
예시: 2D 선형 분류기

VC 차원 = 3

3개 점은 어떤 배치여도 분류 가능:



4개 점은 일부 배치 불가능:



일반화 오차 bound:

$$R(f) \leq \hat{R}(f) + \sqrt{\frac{d(\log(2n/d)+1) - \log(\delta/4)}{n}}$$

여기서:

- d: VC 차원
- n: 데이터 개수
- δ: 신뢰 수준

의미:

$$\text{필요한 데이터} \propto \text{VC 차원}$$

해결 방법:

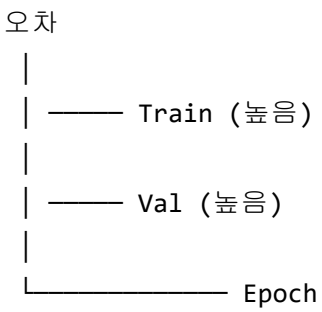
방법	설명	효과
데이터 증가	더 많은 샘플 수집	Variance ↓
정규화	L1, L2, Dropout	수용력 제한
단순화	파라미터 감소	Variance ↓, Bias ↑
Early Stopping	검증 오차 모니터링	과적합 방지
Ensemble	여러 모델 평균	Variance ↓

학습 곡선 해석

학습 곡선:

Epoch에 따른 훈련/검증 오차의 변화

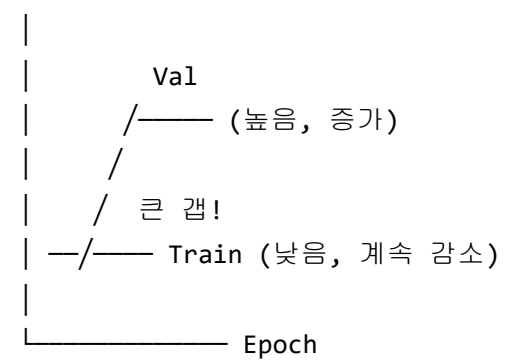
패턴 1: 과소적합



진단:
- 둘 다 높고 평행
- 더 학습해도 개선 안 됨

해결:
- 더 복잡한 모델
- 특징 추가

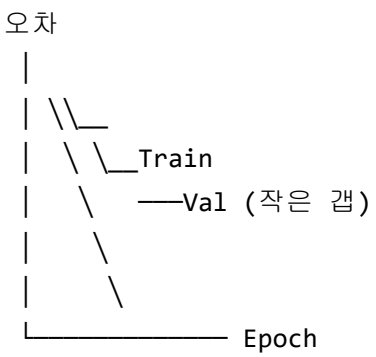
패턴 2: 과적합



진단:
- Train 계속 감소
- Val 증가 또는 정체
- 갭 증가

해결:
- Early stopping
- 정규화
- 데이터 증강

패턴 3: Good Fit

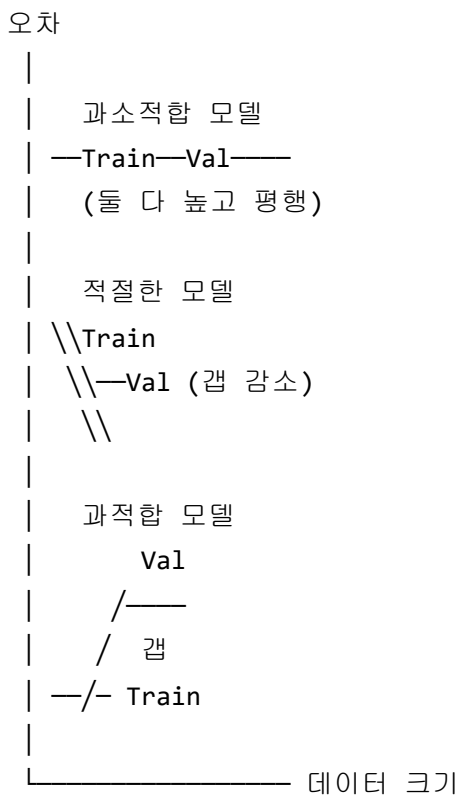


진단:
- 둘 다 낮음
- 갭 작음
- 수렴

결과:
✅ 잘 학습됨!

데이터 크기의 영향

학습 곡선 (vs 데이터 크기):



→ 데이터 많으면 갭 줄어듦!

통찰:

> **💡 데이터는 최고의 정규화**
>
> 많은 데이터는 과적합을 자연스럽게 방지합니다!

4장 요약 정리 📝

핵심 개념

1. 수용력 (Capacity)

```
정의: 모델이 표현할 수 있는 함수의 범위
영향 요소: 파라미터 수, 구조, 정규화
원칙: 데이터 복잡도에 맞춰야 함

필요한 데이터 ∝ 수용력
```

2. 편향-분산 트레이드오프

공식: Total Error = Bias² + Variance + Irreducible Error

Bias: 모델이 평균적으로 틀린 정도
- 원인: 모델이 단순, 가정이 틀림
- 과소적합의 주범

Variance: 학습 데이터에 따라 예측이 변하는 정도
- 원인: 모델이 복잡, 데이터 부족
- 과적합의 주범

트레이드오프: 하나 줄이면 다른 하나 증가
최적점: Bias²과 Variance의 균형

3. 과적합/과소적합 심화

	과소적합	Good Fit	과적합
수용력	부족	적절	과다
Bias	높음	낮음	매우 낮음
Variance	낮음	중간	높음
Train Error	높음	낮음	매우 낮음
Test Error	높음	낮음	높음
일반화 갭	작음	작음	큼

실전 가이드

과소적합 진단:

✓ 훈련 오차 높음
✓ 검증 오차 높음
✓ 둘의 차이 작음

해결:
1. 더 복잡한 모델
2. 특징 추가
3. 정규화 감소
4. 더 오래 학습

과적합 진단:

✓ 훈련 오차 매우 낮음
✓ 검증 오차 높음
✓ 큰 일반화 갭

해결:
1. 더 많은 데이터
2. 정규화 (L1, L2, Dropout)
3. Early stopping
4. 단순한 모델
5. Ensemble

핵심 질문과 답

Q1: 복잡한 모델이 항상 더 좋은가?

A: No! 데이터가 적으면 과적합.
수용력은 데이터 크기에 맞춰야 함.

Q2: Bias와 Variance를 동시에 줄일 수 있나?

A: 일반적으로 No (트레이드오프).
하지만:
- 더 많은 데이터 → 둘 다 감소
- Ensemble → Variance만 감소

Q3: 언제 학습을 멈춰야 하나?

```
A: Early stopping:
   검증 오차가 증가하기 시작하면 중단
```

Q4: VC 차원의 실용적 의미는?

```
A: 필요한 데이터 ≈ VC 차원의 10배 이상
   (경험적 규칙)
```

다음 챕터 예고 ➡️SOON

Chapter 5에서는:

- **성능 평가 실무**
- 데이터 분할 전략 (Hold-out, K-fold, Stratified)
- 분류 평가지표 (Precision, Recall, F1, ROC, PR)
- 회귀 평가지표 (MAE, MSE, R^2)
- 실전: 어떤 지표를 언제 써야 하나?

연결:

```
Chapter 3: 학습의 품질 (현상 이해)
         ↓
Chapter 4: 모델의 근본 원리 (이론 깊이 파기)
         ↓
Chapter 5: 성능 평가 실무 (실전 적용)
```

🎓 복습 문제

1. 수용력을 결정하는 3가지 요소는?
2. Bias와 Variance를 각각 한 문장으로 설명하라.
3. 편향-분산 트레이드오프가 존재하는 이유는?
4. 과소적합과 과적합을 학습 곡선으로 구별하는 방법은?
5. VC 차원이 높으면 무슨 문제가 생기나?

Chapter 5: 성능 평가 실무 📏

들어가며

Chapter 3과 4에서 우리는 **이론**을 배웠습니다:

- 일반화 오차와 훈련 오차의 차이
- 편향-분산 트레이드오프
- 과적합/과소적합의 원리

이제 **실무**로 들어갑니다! 실제로 모델을 평가하고 비교하는 방법을 배웁니다.

이번 챕터의 핵심 질문:

> 🤔 "데이터를 어떻게 나눠야 공정하게 평가할까?"
>
> 🤔 "Accuracy 99%면 좋은 모델인가?"
>
> 🤔 "어떤 평가지표를 언제 써야 하나?"

5.1 데이터 분할 전략 🔀

왜 데이터를 나눠야 하는가?

잘못된 접근:

```
전체 데이터로 학습 → 같은 데이터로 평가

문제: "자기가 본 문제로 자기를 평가" → 부정행위!
      훈련 오차만 측정 (일반화 오차 모름)
```

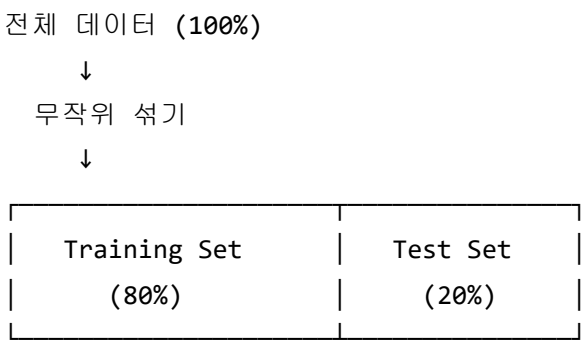
올바른 접근:

```
데이터를 나눔:
- 학습용: 모델 학습
- 평가용: 모델 성능 측정 (안 본 데이터!)

→ 일반화 오차를 근사적으로 측정 ✅
```

5.1.1 Hold-out 방법

가장 단순한 방법:



절차:

1. 전체 데이터를 무작위로 섞음
2. 80%를 학습용으로 분리
3. 20%를 테스트용으로 분리
4. 학습용으로 모델 학습
5. 테스트용으로 성능 평가

분할 비율:

- 보통 **70:30**, **80:20**, **90:10**
- 데이터 많으면: 90:10
- 데이터 적으면: 70:30

장점:

- ✅ 간단하고 빠름
- ✅ 구현 쉬움

단점:

- ❌ 운이 나쁘면 편향된 분할
- ❌ 데이터 적으면 불안정
- ❌ 테스트 데이터를 학습에 못 씀 (낭비)

5.1.2 K-Fold 교차 검증 (Cross-Validation)

Hold-out의 단점 해결!

```
전체 데이터를 K개로 나눔 (예: K=5)

┌─────┬─────┬─────┬─────┬─────┐
│  1  │  2  │  3  │  4  │  5  │
└─────┴─────┴─────┴─────┴─────┘

Fold 1: [Test][Train Train Train Train] → Score₁
Fold 2: [Train][Test][Train Train Train] → Score₂
Fold 3: [Train Train][Test][Train Train] → Score₃
Fold 4: [Train Train Train][Test][Train] → Score₄
Fold 5: [Train Train Train Train][Test] → Score₅

최종 성능 = (Score₁ + Score₂ + ... + Score₅) / 5
```

절차:

1. 데이터를 K개 fold로 나눔
2. For i = 1 to K:
   a. i번째 fold를 테스트로
   b. 나머지 K-1개를 학습으로
   c. 모델 학습 및 평가 → Score_i
3. K개 점수의 평균

K 선택:

- **K=5**: 일반적, 균형잡힘
- **K=10**: 더 정확, 시간 오래 걸림
- **K=n (LOOCV)**: Leave-One-Out, 데이터 극소량

장점:

- ✅ **모든 데이터 활용** (학습 + 테스트 둘 다)
- ✅ 안정적인 평가
- ✅ 평균과 분산 제공

단점:

- ❌ K배 느림 (K번 학습)
- ❌ 계산 비용 높음

예시:

```python
from sklearn.model_selection import cross_val_score

# 5-Fold 교차 검증
scores = cross_val_score(model, X, y, cv=5)

print(f"각 Fold 점수: {scores}")
print(f"평균 ± 표준편차: {scores.mean():.3f} ± {scores.std():.3f}")

# 출력 예:
# 각 Fold 점수: [0.82, 0.85, 0.79, 0.88, 0.84]
# 평균 ± 표준편차: 0.836 ± 0.031
```

5.1.3 Stratified K-Fold

문제: 불균형 데이터에서 K-Fold가 불공정할 수 있음

```
전체 데이터: 양성 10%, 음성 90%

일반 K-Fold:
Fold 1: 양성 15%, 음성 85% (운 좋음)
Fold 2: 양성  5%, 음성 95% (운 나쁨)
→ 불공정!
```

해결: Stratified K-Fold

> 각 Fold에서 **클래스 비율을 원본과 동일하게** 유지

```
전체: 양성 10%, 음성 90%

Stratified K-Fold:
Fold 1: 양성 10%, 음성 90% ✅
Fold 2: 양성 10%, 음성 90% ✅
...
→ 공정!
```

언제 사용:

- 분류 문제
- 불균형 데이터 (클래스 비율이 치우침)
- 작은 데이터셋

예시:

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5)
scores = cross_val_score(model, X, y, cv=skf)
```

5.1.4 시계열 데이터의 특수성

문제: 시계열은 순서가 중요!

잘못된 방법:

```
무작위 섞기:
[1월, 2월, 3월, 4월, 5월, 6월]
      ↓ shuffle
[3월, 6월, 1월, 5월, 2월, 4월]

문제: 미래(6월)로 과거(5월) 예측? → 정보 유출!
```

올바른 방법: Time Series Split

```
시간 순서 유지:

Split 1: [Train      ][Test]
Split 2: [Train Train][Test]
Split 3: [Train Train Train][Test]

→ 과거로 미래를 예측 (현실적!)
```

예시:

```
from sklearn.model_selection import TimeSeriesSplit

tscv = TimeSeriesSplit(n_splits=5)

for train_idx, test_idx in tscv.split(X):
    X_train, X_test = X[train_idx], X[test_idx]
    # 학습 및 평가
```

5.1.5 Train / Validation / Test 3-way Split

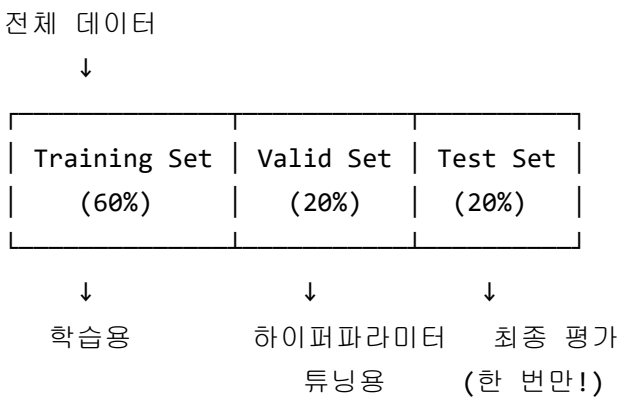
문제: 하이퍼파라미터를 어떻게 선택하나?

잘못된 접근:

```
1. Train으로 학습
2. Test로 여러 하이퍼파라미터 시도
3. 가장 좋은 것 선택

문제: Test에 과적합! (Test를 "보면서" 조정)
```

올바른 접근: 3-way split



각 데이터셋의 역할:

데이터셋	역할	예시
Training	파라미터 학습	가중치 w, b
Validation	하이퍼파라미터 선택	학습률, 정규화 강도
Test	최종 성능 평가	논문 보고 성능

워크플로우:

```
1. 데이터를 Train / Val / Test로 분할

2. 하이퍼파라미터 탐색:
   For each 하이퍼파라미터:
       Train으로 학습
       Val으로 평가 → 점수 기록

3. 최고 점수의 하이퍼파라미터 선택

4. Train+Val로 최종 모델 재학습 (선택적)

5. Test로 최종 평가 (한 번만!)
```

> ⚠️ **중요 원칙**
>
> **Test Set은 절대로 "보지" 말 것!**
>
> - Test로 모델 조정 → Test에 과적합
> - Test는 최종 평가에만 한 번만 사용

5.1.6 부트스트랩 (Bootstrap)

부트스트랩이란?

> 📘 **복원 추출**로 원본 데이터와 같은 크기의 샘플을 여러 번 생성하는 방법

왜 부트스트랩이 필요한가?

문제 상황:

```
모델 성능: 정확도 85%

질문: 이게 운이 좋아서 85%? 실력으로 85%?
      만약 다른 데이터로 학습했다면 80%? 90%?
      → 불확실성을 어떻게 측정?
```

일반적 해결책:

- 여러 번 실험 (데이터 새로 수집 → 학습 → 평가)
- 문제: 데이터 수집 비용 큼! ❌

부트스트랩 해결책:

- 기존 데이터로 "가상의 여러 데이터셋" 생성!
- 비용 0! ✅

> 💡 **"복원 추출"이 뭔가요?**
>
> **생활 속 비유: 제비뽑기**
>
> **비복원 추출 (일반적):**
>
> ```
> 상자: [빨강, 파랑, 초록, 노랑, 보라] (5개)
>
> 3개 뽑기:
>   1. 빨강 뽑음 → 빨강 제거 (상자에 4개 남음)
>   2. 파랑 뽑음 → 파랑 제거 (상자에 3개 남음)
>   3. 초록 뽑음 → 초록 제거 (상자에 2개 남음)
>
> 결과: [빨강, 파랑, 초록]
> → 각 색이 최대 1번만 나옴
> ```
>
> **복원 추출 (Bootstrap):**
>
> ```
> 상자: [빨강, 파랑, 초록, 노랑, 보라] (5개)
>
> 3개 뽑기:
>   1. 빨강 뽑음 → 다시 넣음! (상자에 여전히 5개)
>   2. 빨강 뽑음 → 다시 넣음! (또 빨강 가능!)
>   3. 초록 뽑음 → 다시 넣음!
>
> 결과: [빨강, 빨강, 초록]
> → 같은 색이 여러 번 나올 수 있음!
> → 어떤 색은 한 번도 안 나올 수 있음! (파랑, 노랑, 보라)
> ```
>
> **핵심 차이:**
>
> - 비복원: 뽑으면 제거 → 중복 없음
> - **복원: 뽑고 다시 넣음 → 중복 가능!**

절차:

```
원본 데이터: [1, 2, 3, 4, 5] (n=5개)

복원 추출로 5개씩 뽑기:
Bootstrap 1: [1, 2, 2, 4, 5] (3이 안 뽑힘)
  → 뽑기: 1 (넣고) → 2 (넣고) → 2 (또!) → 4 (넣고) → 5
Bootstrap 2: [1, 1, 3, 4, 5] (2가 안 뽑힘)
  → 뽑기: 1 (넣고) → 1 (또!) → 3 (넣고) → 4 (넣고) → 5
Bootstrap 3: [2, 3, 3, 3, 5] (1, 4가 안 뽑힘)
  → 뽑기: 2 (넣고) → 3 (넣고) → 3 (또!) → 3 (또!) → 5
...
Bootstrap B: [1, 2, 4, 4, 4] (3, 5가 안 뽑힘)

→ B개의 부트스트랩 샘플 생성 (보통 B = 100 ~ 1000)
```

구체적 예시:

```
원본 데이터: 100개

부트스트랩 샘플 1번째 생성:
  → 100번 뽑기 (매번 다시 넣으면서)
  → 어떤 데이터는 3번 뽑힐 수도 (중복)
  → 어떤 데이터는 0번 뽑힐 수도 (제외)
  → 결과: 100개 (중복 포함)

부트스트랩 샘플 2번째 생성:
  → 다시 100번 뽑기 (처음부터 다시!)
  → 1번째와 다른 결과
  → 결과: 100개 (다른 중복 패턴)

→ 이렇게 100~1000번 반복!
```

특징:

- **복원 추출**: 같은 데이터가 여러 번 선택될 수 있음 (중복 허용)
- **Out-of-Bag (OOB)**: 각 샘플에서 선택되지 않은 데이터 (~36.8%)

수학적 배경:

하나의 데이터가 **한 번도 선택되지 않을** 확률:

$$P(\text{선택 안 됨}) = \left(1 - \frac{1}{n}\right)^n \xrightarrow{n \to \infty} \frac{1}{e} \approx 0.368$$

→ 약 **36.8%의 데이터**가 각 부트스트랩 샘플에서 제외됨!

장점:

- ✅ **모델 안정성 평가**: 여러 샘플로 분산 측정
- ✅ **신뢰구간 추정**: 성능의 불확실성 파악
- ✅ **작은 데이터셋**: 데이터 부족 시 유용

단점:

- ❌ **계산 비용**: B개 모델 학습 필요
- ❌ **낙관적 편향**: 같은 데이터 중복으로 약간 낙관적

Out-of-Bag (OOB) 평가:

```
각 부트스트랩 샘플에서:
- 선택된 데이터 (~63%) → 학습
- 선택 안 된 데이터 (~37%) → 검증

최종 OOB 성능 = 각 데이터를 포함하지 않은
                모델들로 평가한 평균
```

예시:

```python
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier

# 부트스트랩 샘플 생성
n_bootstraps = 100
scores = []

for i in range(n_bootstraps):
    # 복원 추출
    X_boot, y_boot = resample(X, y, n_samples=len(X))

    # 모델 학습
    model = RandomForestClassifier()
    model.fit(X_boot, y_boot)

    # OOB 데이터로 평가
    # (실제로는 더 복잡한 로직 필요)
    score = model.score(X_test, y_test)
    scores.append(score)

# 성능의 평균과 신뢰구간
print(f"평균 성능: {np.mean(scores):.3f}")
print(f"95% 신뢰구간: [{np.percentile(scores, 2.5):.3f}, "
      f"{np.percentile(scores, 97.5):.3f}]")
```

언제 사용?

상황	설명
앙상블 학습	Random Forest, Bagging
불확실성 추정	신뢰구간 계산
작은 데이터셋	데이터 부족 시
모델 안정성	성능 분산 확인

> 💡 **Random Forest는 부트스트랩 기반!**
>
> 각 트리마다 부트스트랩 샘플로 학습하고, OOB 데이터로 성능 평가

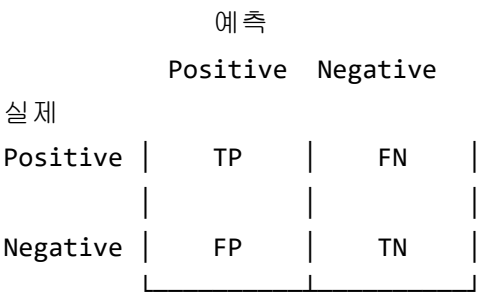
데이터 분할 전략 선택 가이드

상황	추천 방법	이유
데이터 많음	Hold-out	빠르고 충분히 정확
데이터 적음	K-Fold CV	모든 데이터 활용
불균형 데이터	Stratified K-Fold	공정한 평가
시계열	Time Series Split	시간 순서 유지
하이퍼파라미터 튜닝	3-way split	Test 오염 방지
앙상블/불확실성	Bootstrap	신뢰구간 추정

5.2 분류 문제 평가지표 📊

5.2.1 혼동 행렬 (Confusion Matrix)

모든 분류 지표의 기초!



4가지 경우:

약어	전체 이름	의미	예시 (암 진단)
TP	True Positive	실제 O, 예측 O ✅	암을 암으로 진단
TN	True Negative	실제 X, 예측 X ✅	정상을 정상으로 진단
FP	False Positive	실제 X, 예측 O ❌	정상을 암으로 오진
FN	False Negative	실제 O, 예측 X ❌	암을 정상으로 오진

예시: 스팸 필터

100개 이메일:

- 실제 스팸: 15개
- 실제 정상: 85개

예측 결과:

```
              예측: 스팸   예측: 정상
실제 스팸        12           3       ← TP=12, FN=3
실제 정상         5          80       ← FP=5, TN=80
```

시각화: Confusion Matrix

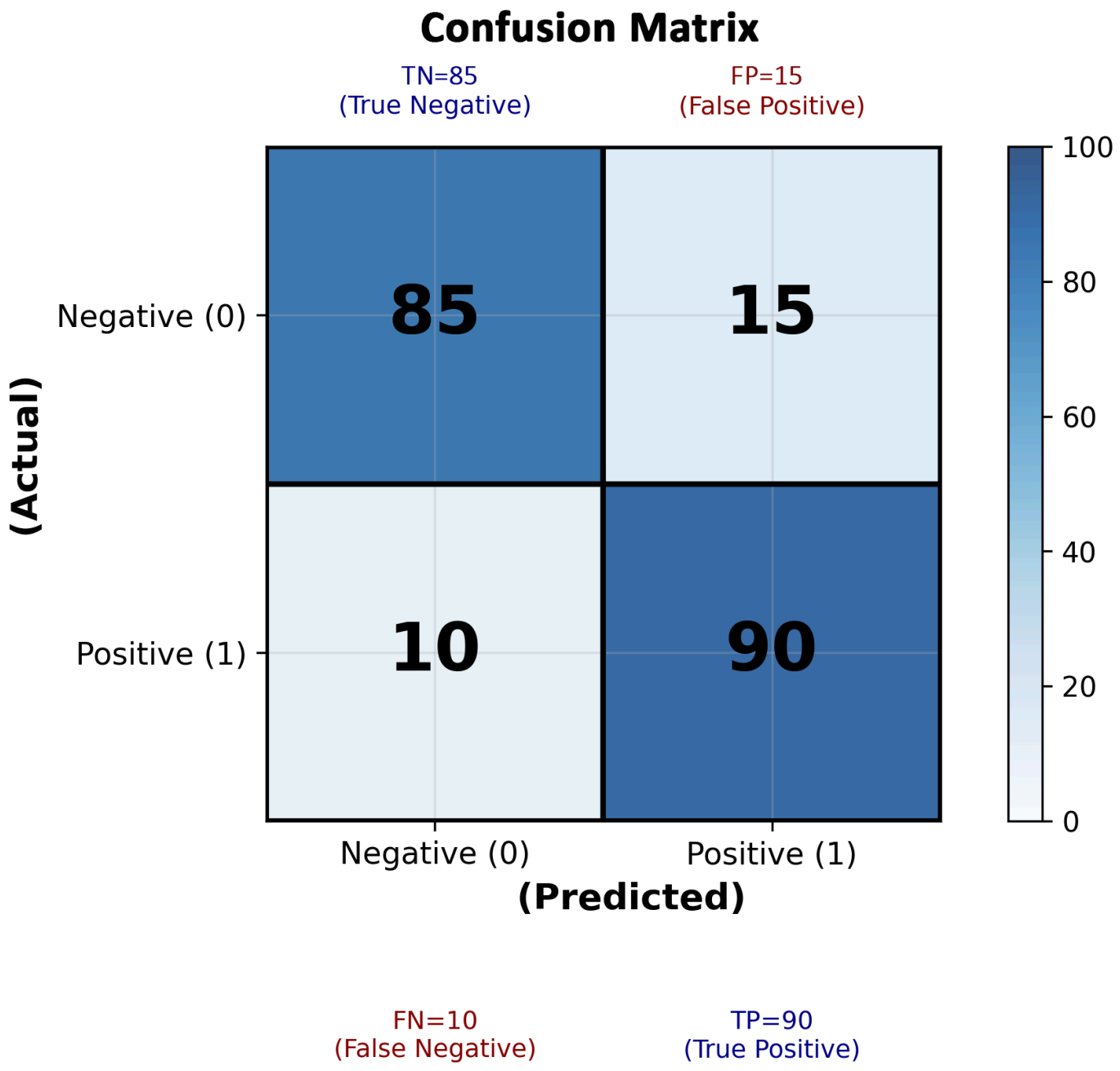


그림 설명: 이진 분류의 Confusion Matrix 예시.

- 왼쪽 위 (TN=85): 정답을 정답으로 예측 (True Negative)
- 오른쪽 위 (FP=15): 거짓을 참으로 잘못 예측 (False Positive)
- 왼쪽 아래 (FN=10): 참을 거짓으로 잘못 예측 (False Negative)
- 오른쪽 아래 (TP=90): 참을 참으로 예측 (True Positive)

5.2.2 기본 지표

1 정확도 (Accuracy)

정의:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

의미: 전체 중 맞춘 비율

예시:

$$\text{Accuracy} = \frac{12 + 80}{100} = 0.92 = 92\%$$

장점:

- ✅ 직관적
- ✅ 계산 간단

단점:

- ❌ **불균형 데이터에서 오해의 소지**

치명적 예시:

```
암 진단:
- 정상: 990명
- 암: 10명 (1%)

"바보 모델": 무조건 "정상" 예측

Accuracy = 990/1000 = 99% 😱

하지만 암 환자를 0명 찾음 → 쓸모없음!
```

2 정밀도 (Precision)

정의:

$$\text{Precision} = \frac{TP}{TP + FP}$$

질문: "Positive라고 예측한 것 중 실제로 Positive인 비율?"

예시:

$$\text{Precision} = \frac{12}{12 + 5} = \frac{12}{17} \approx 0.71 = 71\%$$

해석: "스팸으로 분류한 메일 중 71%가 진짜 스팸"

언제 중요: FP 비용이 클 때!

예시들:

1. **스팸 필터**
   - FP: 중요 메일을 스팸으로 ← 큰 문제!
   - FN: 스팸을 정상으로 ← 조금 귀찮음
2. **유튜브 추천**
   - FP: 안 볼 영상 추천 ← 건너뛰면 됨
   - 정밀도 낮으면 사용자 신뢰 하락

> 💡 **Precision 높다 = 예측 신중 = FP 적음**

3 재현율 (Recall, Sensitivity)

정의:

$$\text{Recall} = \frac{TP}{TP + FN}$$

질문: "실제 Positive 중 모델이 찾아낸 비율?"

예시:

$$\text{Recall} = \frac{12}{12 + 3} = \frac{12}{15} = 0.80 = 80\%$$

해석: "실제 스팸의 80%를 잡아냄"

언제 중요: FN 비용이 클 때!

예시들:

1. **암 진단**
   - FN: 암을 못 찾음 ← 생명 위험! 😱😱
   - FP: 정상을 암으로 오진 ← 재검사 가능
2. **사기 탐지**
   - FN: 사기를 못 잡음 ← 금전 손실!
   - FP: 정상을 사기로 오판 ← 불편

> 💡 **Recall 높다 = 놓치지 않음 = FN 적음**

5.2.3 Precision vs Recall 트레이드오프

왜 트레이드오프가 발생하는가?

핵심 이유: 대부분의 모델은 **확률**을 출력!

```
예: 암 진단 모델

환자 A: 암 확률 95% → "암"으로 분류?
환자 B: 암 확률 70% → "암"으로 분류?
환자 C: 암 확률 45% → "암"으로 분류?
환자 D: 암 확률 20% → "암"으로 분류?

어디서 선을 그을 것인가? → Threshold (임계값)
```

> **💡 Threshold (임계값) 개념**
>
> **정의:** "이 확률 이상이면 Positive로 분류"하는 기준점
>
> **예시:**
>
> ```
> Threshold = 50% (기본값)
>   → 50% 이상: Positive
>   → 50% 미만: Negative
>
> Threshold = 80% (높게 설정)
>   → 80% 이상: Positive (매우 확실할 때만!)
>   → 80% 미만: Negative
>
> Threshold = 30% (낮게 설정)
>   → 30% 이상: Positive (의심만 되어도!)
>   → 30% 미만: Negative
> ```
>
> **구체적 예시:**

```
환자들의 암 확률:
  A: 95%, B: 70%, C: 45%, D: 20%

Threshold = 80%:
  → Positive: A만 (1명)
  → Negative: B, C, D (3명)

Threshold = 50%:
  → Positive: A, B (2명)
  → Negative: C, D (2명)

Threshold = 30%:
  → Positive: A, B, C (3명)
  → Negative: D (1명)
```

문제: 둘을 동시에 높이기 어려움!

Threshold에 따른 변화:

```
Threshold 높임 (80% → 확실할 때만 Positive):
  → Positive 예측 개수 ↓
  → TP 감소, FP 감소
  → Precision ↑ (Positive 예측이 정확해짐)
  → Recall ↓ (많은 Positive를 놓침)

Threshold 낮춤 (30% → 의심만 되어도 Positive):
  → Positive 예측 개수 ↑
  → TP 증가, FP 증가
  → Precision ↓ (오탐이 많아짐)
  → Recall ↑ (거의 모든 Positive를 잡음)
```

예시: 암 진단

전략 A: 보수적 (신중) - Threshold 높음 (80%)

확실할 때만 "암" 진단

Precision: 높음 (진단하면 거의 맞음) ✅
Recall: 낮음 (많은 암 환자 놓침) ❌

구체적:
  실제 암 환자 10명 중 3명만 찾음 (Recall 30%)
  하지만 "암"이라 진단한 3명은 모두 진짜 (Precision 100%)

전략 B: 공격적 (민감) - Threshold 낮음 (30%)

조금만 의심되어도 "암" 진단

Precision: 낮음 (오진 많음) ❌
Recall: 높음 (거의 모든 암 찾음) ✅

구체적:
  실제 암 환자 10명 중 9명 찾음 (Recall 90%)
  하지만 정상 환자 20명도 "암"으로 오진 (Precision 31%)

시각화:

결정 임계값 변화:

임계값 0.9 (보수적)
  → Precision ↑, Recall ↓

임계값 0.5 (균형)
  → Precision 중간, Recall 중간

임계값 0.1 (공격적)
  → Precision ↓, Recall ↑

5.2.4 F1 Score

문제: Precision과 Recall 중 뭘 우선?

해결: 둘의 조화 평균!

$$F_1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

왜 조화 평균?

```
Precision = 90%, Recall = 10%

산술 평균 = (90 + 10) / 2 = 50% ← 오해!

F1 (조화) = 2 × (90×10)/(90+10) = 18% ← 현실적!
```

특성:

- 두 값이 모두 높아야 높음
- 한 쪽이 낮으면 전체가 낮음
- **균형** 강조

예시:

Precision	Recall	F1 Score
90%	90%	90%
90%	10%	18% ⚠
50%	50%	50%
100%	10%	18% ⚠

F-beta Score (일반화):

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \times \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

- $\beta = 1$: F1 (균형)
- $\beta = 0.5$: Precision 중시
- $\beta = 2$: Recall 중시

사용 예:

- 암 진단: F_2 (Recall 2배 중시)
- 스팸 필터: $F_{0.5}$ (Precision 중시)

5.2.5 ROC 곡선과 AUC

문제: 임계값에 따라 Precision/Recall 변함

해결: 모든 임계값을 고려!

ROC 곡선 (Receiver Operating Characteristic):

- x축: FPR = $\frac{FP}{TN+FP}$ (False Positive Rate)
- y축: TPR = $\frac{TP}{TP+FN}$ = Recall

시각화: ROC Curve

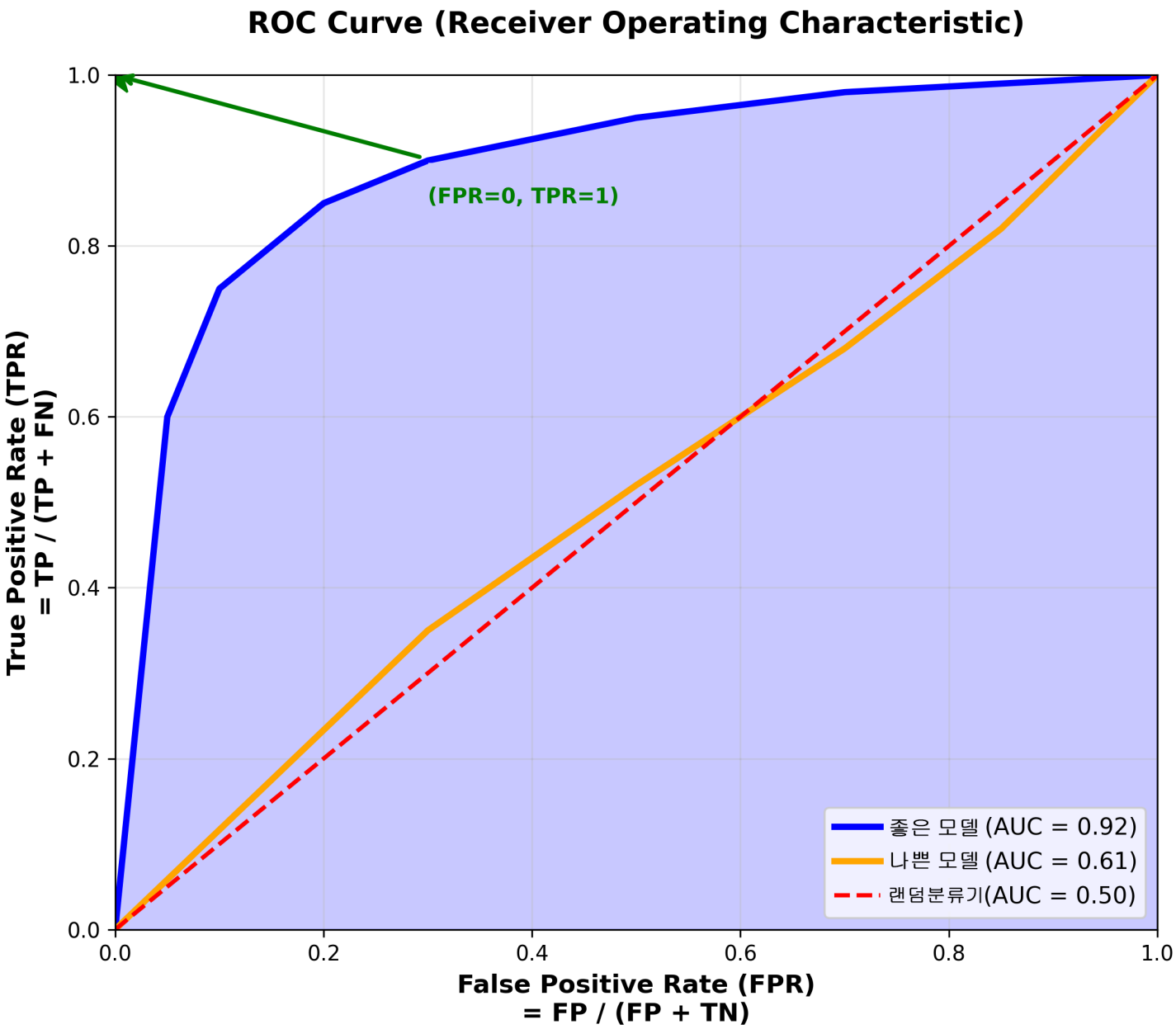


그림 설명:

- **파란 곡선 (좋은 모델)**: AUC = 0.92로 왼쪽 위에 가깝게 위치. 낮은 FPR에서 높은 TPR을 달성합니다.
- **주황 곡선 (나쁜 모델)**: AUC = 0.61로 대각선에 가깝습니다.
- **빨간 대각선**: 랜덤 분류기 (AUC = 0.50). 동전 던지기와 같은 성능.

- **녹색 별 (0, 1)**: 완벽한 모델의 위치. 오탐 없이 모든 Positive를 잡습니다.

해석:

완벽한 모델:

```
(0, 1) ← FPR=0, TPR=1
"오탐 없이 모두 찾음!"
```

무작위:

```
대각선
"동전 던지기와 같음"
```

AUC (Area Under Curve):

ROC 곡선 아래 면적

AUC	의미
1.0	완벽 ✅
0.9~1.0	매우 좋음
0.8~0.9	좋음
0.7~0.8	괜찮음
0.5~0.7	나쁨
0.5	무작위
<0.5	무작위보다 못함 😱

장점:

- ✅ 임계값 독립적
- ✅ 불균형 데이터에 robust
- ✅ 모델 비교 용이

단점:

- ❌ 어떤 임계값 쓸지 안 알려줌
- ❌ FP/FN 비용 다를 때 부적합

5.2.6 PR 곡선 (Precision-Recall Curve)

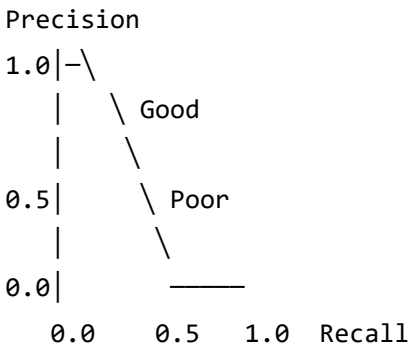
문제: ROC는 불균형 데이터에서 낙관적

```
데이터: 정상 99%, 이상 1%

나쁜 모델도 TN 많아서 FPR 낮음
→ ROC가 좋게 보임
```

해결: PR 곡선

- x축: Recall
- y축: Precision



ROC vs PR:

	ROC	PR
x축	FPR (TN 포함)	Recall (TN 미포함)
불균형	낙관적	현실적 ✅
사용	균형 데이터	불균형 데이터

AP (Average Precision):

PR 곡선 아래 면적 (AUC의 PR 버전)

5.2.7 불균형 데이터 평가

불균형 데이터:

클래스 분포가 치우침

예:
- 사기: 0.1%, 정상: 99.9%
- 암: 1%, 정상: 99%
- 스팸: 5%, 정상: 95%

잘못된 평가:

```
"바보 모델":
def predict(x):
    return "정상"  # 항상 다수 클래스

Accuracy = 99% 😱
→ 쓸모없는데 점수 높음!
```

올바른 평가 전략:

1. Accuracy 쓰지 말 것!

대신:

- Precision, Recall, F1
- ROC AUC (균형 데이터)
- PR AUC (불균형 데이터) ✅

2. 혼동 행렬 자세히 보기

```
            예측
          정상    암
실제
정상    | 990  |  0  |
        |      |     |
암      |  10  |  0  |
        └──────┴─────┘

Accuracy = 99%
하지만 암 Recall = 0% ← 최악!
```

3. 클래스별 평가

```
정상 클래스:
- Precision: 990/1000 = 99%
- Recall: 990/990 = 100%

암 클래스:
- Precision: 0/0 = undefined
- Recall: 0/10 = 0% ← 문제!
```

4. Macro vs Weighted Average

Macro (클래스별 평균):

$$\text{Macro-F1} = \frac{F1_{\text{정상}} + F1_{\text{암}}}{2}$$

모든 클래스를 **동등하게** 취급

Weighted (샘플 수로 가중):

$$\text{Weighted-F1} = \frac{n_{\text{정상}} \cdot F1_{\text{정상}} + n_{\text{암}} \cdot F1_{\text{암}}}{n_{\text{total}}}$$

많은 클래스에 **더 큰 가중치**

예시:

```
정상 (990개): F1 = 99.5%
암 (10개): F1 = 0%

Macro-F1 = (99.5 + 0) / 2 = 49.75%
Weighted-F1 = (990×99.5 + 10×0) / 1000 = 98.5%

→ Macro가 불균형을 더 잘 반영 ✅
```

5.3 회귀 문제 평가지표 📈

5.3.1 평균 절대 오차 (MAE)

정의:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

예시: 집값 예측

실제	예측	오차	절대 오차
3.0억	2.8억	-0.2억	0.2억
5.0억	5.3억	+0.3억	0.3억
4.0억	3.9억	-0.1억	0.1억

$$\text{MAE} = \frac{0.2 + 0.3 + 0.1}{3} = 0.2\text{억}$$

해석: "평균적으로 2천만원 틀림"

장점:

- ✅ 직관적
- ✅ 원래 단위
- ✅ 이상치에 강건

단점:

- ❌ 0에서 미분 불가

5.3.2 평균 제곱 오차 (MSE, RMSE)

MSE:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

RMSE:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

같은 예시:

오차	제곱 오차
-0.2억	0.04억²

오차	제곱 오차
+0.3억	0.09억²
-0.1억	0.01억²

$$\text{MSE} = \frac{0.04 + 0.09 + 0.01}{3} = 0.047\text{억}^2$$

$$\text{RMSE} = \sqrt{0.047} \approx 0.22\text{억}$$

장점:

- ✅ 미분 가능
- ✅ 큰 오차에 페널티

단점:

- ❌ 이상치에 민감
- ❌ 단위가 제곱 (MSE)

MAE vs RMSE:

```
이상치 영향:

데이터: [1, 2, 3, 100]
예측:   [1, 2, 3,  3]
오차:   [0, 0, 0, 97]

MAE  = 97/4 = 24.25
RMSE = √(97²/4) = 48.5

→ RMSE가 2배 큰 영향!
```

5.3.3 R² (결정계수)

문제: MAE/MSE는 **척도 의존적**

```
모델 A: MAE = 1000 (집값, 단위: 억)
모델 B: MAE = 10 (온도, 단위: ℃)

Q: A가 B보다 100배 나쁜가?
A: 알 수 없음! (단위가 다름)
```

해결: R²

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

의미:

Baseline: 평균으로 예측

```
모델: f(x) = ȳ (상수)
오차 = SS_tot
```

우리 모델:

```
오차 = SS_res

R² = 1 - (우리 모델 오차) / (Baseline 오차)
```

해석:

R²	의미
1.0	완벽 ✅
0.9~1.0	매우 좋음
0.7~0.9	좋음
0.5~0.7	괜찮음
0.0~0.5	나쁨
0.0	Baseline과 같음
<0.0	Baseline보다 못함 😱

장점:

- ✅ 척도 독립적
- ✅ 해석 쉬움 (0~1)
- ✅ 모델 비교 용이

단점:

- ❌ 특징 추가하면 무조건 증가
- ❌ 외삽(extrapolation)에서 오해

Adjusted R²:

특징 추가에 페널티:

$$\text{Adj-}R^2 = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$$

여기서:

- n: 샘플 수
- p: 특징 수

5.4 실전 가이드: 어떤 지표를 언제? 🎯

5.4.1 분류 문제 지표 선택

```
┌─ 데이터 균형적?
│
├─ Yes → Accuracy 사용 가능
│         (+ Precision, Recall 참고)
│
└─ No (불균형) →
   │
   ├─ FP 비용 큼? (예: 스팸)
   │  → Precision 중시, F0.5
   │
   ├─ FN 비용 큼? (예: 암 진단)
   │  → Recall 중시, F2
   │
   ├─ 균형 필요?
   │  → F1
   │
   └─ 임계값 독립 평가?
      ├─ 균형 데이터 → ROC AUC
      └─ 불균형 데이터 → PR AUC
```

상황별 추천:

상황	추천 지표	이유
균형 데이터	Accuracy, F1	직관적
불균형 (FP 나쁨)	Precision, F0.5	FP 최소화
불균형 (FN 나쁨)	Recall, F2	FN 최소화
불균형 (균형)	F1	절충
모델 비교 (균형)	ROC AUC	임계값 독립
모델 비교 (불균형)	PR AUC	현실적

5.4.2 회귀 문제 지표 선택

```
┌─ 이상치 많음?
│
├─ Yes → MAE 또는 Huber Loss
│         (강건함)
│
└─ No →
   │
   ├─ 서로 다른 문제 비교?
   │  → R² 또는 MAPE
   │      (척도 독립)
   │
   ├─ 큰 오차에 페널티?
   │  → RMSE
   │
   └─ 일반적 상황
      → RMSE (표준)
```

상황별 추천:

상황	추천 지표	이유
일반적	RMSE	표준, 미분 가능
이상치 많음	MAE	강건
척도 비교	R^2, MAPE	독립적
0 값 있음	MAE, RMSE	MAPE 불가
백분율 중요	MAPE	상대 오차

5.4.3 실전 체크리스트

평가 전:

```
[ ] 데이터 분할 완료? (Train/Val/Test)
[ ] Test는 안 봤는가?
[ ] 불균형 확인? (Stratified 필요?)
[ ] 시계열? (Time Series Split 필요?)
```

평가 중:

```
[ ] 여러 지표 확인? (하나만 보지 말 것)
[ ] 혼동 행렬 확인? (분류)
[ ] 클래스별 성능? (불균형)
[ ] 학습 곡선 확인? (과적합 진단)
```

평가 후:

```
[ ] 교차 검증 완료?
[ ] 표준편차 확인? (안정성)
[ ] Test 성능 < Val 성능? (정상)
[ ] Test 성능 >> Val 성능? (의심!)
```

5장 요약 정리 📝

데이터 분할

방법	장점	단점	사용
Hold-out	빠름, 간단	불안정	데이터 많음
K-Fold	안정, 모든 데이터 활용	느림	일반적
Stratified	공정	-	불균형
Time Series	시간 순서 유지	-	시계열

분류 지표

지표	공식	언제 중요
Accuracy	$\frac{TP+TN}{All}$	균형 데이터
Precision	$\frac{TP}{TP+FP}$	FP 비용 큼
Recall	$\frac{TP}{TP+FN}$	FN 비용 큼
F1	$\frac{2PR}{P+R}$	균형 필요

지표	공식	언제 중요
ROC AUC	곡선 아래 면적	임계값 독립

회귀 지표

지표	특징	사용
MAE	직관적, 강건	이상치 多
RMSE	큰 오차 페널티	일반적
R^2	척도 독립, 0~1	모델 비교

핵심 원칙

1. 상황에 맞는 지표 선택

- 문제 유형 (분류/회귀)
- 데이터 균형
- 비용 구조 (FP vs FN)

2. 여러 지표 종합 평가

하나의 지표만 보지 말 것!
다각도로 평가

3. Test는 마지막에 한 번만

Test로 튜닝 금지!
Val로 개발, Test로 최종 평가

🎓 최종 복습 문제

1. K-Fold와 Hold-out의 차이는?
2. Stratified K-Fold는 언제 사용하나?
3. Precision과 Recall의 차이를 예시로 설명하라.
4. 불균형 데이터에서 Accuracy를 쓰면 안 되는 이유는?

5. MAE vs RMSE: 어떤 상황에 각각 적합한가?
6. R^2가 음수일 수 있는가? 의미는?

2차 스터디 전체 정리 🎓

배운 내용:

```
Chapter 1: 왜 머신러닝이 작동하는가?
- 귀납적 편향
- No Free Lunch Theorem
- 학습의 철학적 기초

Chapter 2: 어떻게 학습하는가?
- 손실함수
- 최적화와 경사하강법
- Adam, SGD 등

Chapter 3: 학습의 품질 이해하기
- 훈련 오차 vs 일반화 오차
- 일반화 갭
- 과적합/과소적합 (기본)

Chapter 4: 모델의 근본 원리
- 수용력
- 편향-분산 트레이드오프
- 과적합/과소적합 (심화)

Chapter 5: 성능 평가 실무
- 데이터 분할 전략
- 분류/회귀 평가지표
- 실전 가이드
```

다음 단계:

이제 여러분은 머신러닝의 **이론적 기초**를 탄탄히 다졌습니다!

다음 스터디에서는:

- 구체적인 알고리즘들 (SVM, 결정 트리, 신경망)
- 앙상블 방법 (Bagging, Boosting)

- 차원 축소와 특징 선택
- 실전 프로젝트

여러분의 머신러닝 여정에 행운을 빕니다! 🚀

부록: 용어 사전 (Glossary)

A

Adam (Adaptive Moment Estimation)

- 적응적 학습률을 사용하는 최적화 알고리즘
- 1차 모멘트(평균)와 2차 모멘트(분산)를 모두 활용
- 현대 딥러닝의 기본 옵티마이저

Accuracy (정확도)

- 전체 예측 중 맞춘 비율
- 공식: (TP + TN) / (TP + TN + FP + FN)
- 불균형 데이터에서는 오해의 소지

B

Bias (편향)

- 모델의 평균 예측이 실제 값에서 벗어난 정도
- 높은 Bias = 과소적합 (모델이 너무 단순)
- Variance와 트레이드오프 관계

Bootstrap (부트스트랩)

- 복원 추출로 여러 샘플을 생성하는 방법
- 모델 안정성 평가 및 신뢰구간 추정
- Random Forest의 핵심 기법

C

Capacity (수용력)

- 모델이 표현할 수 있는 함수의 복잡도

- 높은 수용력 = 복잡한 패턴 학습 가능
- 낮은 수용력 = 단순한 패턴만 학습 가능

Cross-Entropy (교차 엔트로피)

- 분류 문제의 손실 함수
- "예측 확률"과 "정답" 사이의 차이 측정
- 로그를 사용하여 "놀람의 정도" 반영

Cross-Validation (교차 검증)

- 데이터를 K개 폴드로 나누어 K번 검증
- 모든 데이터를 학습과 검증에 활용
- 작은 데이터셋에서 유용

E

Expectation (기댓값)

- 확률적 변수의 평균값
- 표기: $\mathbb{E}[\cdot]$
- 예: 주사위 기댓값 = 3.5

F

F1 Score

- Precision과 Recall의 조화 평균
- 공식: $2 \times \frac{P \times R}{P + R}$
- 두 지표의 균형 중시

G

Generalization Error (일반화 오차)

- 학습하지 않은 새 데이터에서의 오차
- 머신러닝의 진짜 목표
- 테스트 세트로 근사 측정

Gradient Descent (경사하강법)

- 손실 함수의 최소값을 찾는 최적화 알고리즘

- 그래디언트의 반대 방향으로 파라미터 업데이트
- 변형: SGD, Mini-batch, Momentum, Adam

Gradient (그래디언트)

- 다변수 함수의 모든 편미분을 모은 벡터
- "가장 빠르게 증가하는 방향" 지시
- 표기: ∇f

H

Hypothesis Space (가설 공간)

- 모델이 학습을 통해 도달할 수 있는 모든 함수들의 집합
- 표기: $\mathcal{H} = \{h_\theta \mid \theta \in \Theta\}$
- 예: 선형 회귀의 가설 공간 = 모든 직선

I

Inductive Bias (귀납적 편향)

- 학습 알고리즘이 가진 선호 또는 가정
- 예: 선형 회귀는 "직선 관계" 선호
- No Free Lunch: 모든 문제에 완벽한 편향은 없음

Irreducible Error (축소 불가능한 오차)

- 데이터 자체의 노이즈
- 어떤 모델로도 줄일 수 없음
- 예: 측정 오차, 랜덤 노이즈

L

Learning Rate (학습률)

- 경사하강법에서 파라미터 업데이트 보폭
- 표기: η (eta)
- 너무 크면 발산, 너무 작으면 느림

Loss Function (손실 함수)

- 모델의 예측과 정답의 차이를 측정

- 회귀: MSE, MAE
- 분류: Cross-Entropy

M

Momentum (모멘텀)

- 과거 그래디언트를 기억하여 관성 추가
- 진동 감소, 수렴 속도 향상
- 공식: $v^{(t+1)} = \gamma v^{(t)} + \eta \nabla L$

N

No Free Lunch Theorem (NFL 정리)

- "모든 문제에 완벽한 알고리즘은 없다"
- 한 문제에 좋은 알고리즘이 다른 문제엔 나쁠 수 있음
- 귀납적 편향의 중요성

O

Overfitting (과적합)

- 훈련 데이터에 과도하게 맞춤
- 훈련 오차는 낮지만 테스트 오차는 높음
- 원인: 모델 너무 복잡, 데이터 너무 적음

Out-of-Bag (OOB)

- 부트스트랩 샘플에서 선택되지 않은 데이터 (~36.8%)
- 검증 세트로 활용 가능

P

Partial Derivative (편미분)

- 다변수 함수에서 한 변수에 대한 미분
- 다른 변수는 고정
- 표기: $\frac{\partial f}{\partial x}$

Precision (정밀도)

- Positive로 예측한 것 중 실제 Positive 비율
- 공식: TP / (TP + FP)
- "예측의 정확성" 측정

R

Recall (재현율, 민감도)

- 실제 Positive 중 모델이 찾아낸 비율
- 공식: TP / (TP + FN)
- "놓치지 않고 찾기" 측정

Regularization (정규화)

- 과적합 방지 기법
- L1, L2: 파라미터 크기에 페널티
- Dropout, Early Stopping 등

S

SGD (Stochastic Gradient Descent)

- 데이터 하나씩 사용하는 경사하강법
- 빠르지만 불안정
- Mini-batch SGD가 실전 표준

T

Training Error (훈련 오차)

- 학습에 사용한 데이터에서의 오차
- 낮다고 좋은 것 아님 (과적합 가능)
- Generalization Error가 진짜 목표

Threshold (임계값)

- 분류 결정을 위한 확률 기준점
- 기본값: 0.5 (50%)
- 조정하여 Precision-Recall 트레이드오프 조절

U

Underfitting (과소적합)

- 모델이 너무 단순하여 패턴을 못 잡음
- 훈련 오차도 높고 테스트 오차도 높음
- 해결: 더 복잡한 모델 사용

V

Validation Set (검증 세트)

- 하이퍼파라미터 튜닝 및 모델 선택용
- 학습에는 사용하지 않음
- 여러 번 사용 가능 (테스트는 한 번만)

Variance (분산)

- 학습 데이터가 바뀔 때 모델 예측의 변동성
- 높은 Variance = 과적합 (모델이 불안정)
- Bias와 트레이드오프 관계

Version Space (버전 공간)

- 훈련 데이터를 완벽히 맞추는 모든 가설들의 집합
- Hypothesis Space의 부분집합
- 학습 = Version Space 축소 과정

부록: 수식 기호 정리

그리스 문자

기호	이름	일반적 용도
α	alpha	학습률, 정규화 계수
β	beta	모멘텀 계수, F-score 가중치
γ	gamma	모멘텀 계수, 할인율

기호	이름	일반적 용도
δ	delta	변화량, 오차
ϵ	epsilon	노이즈, 작은 수
η	eta	학습률
θ	theta	파라미터 (일반)
λ	lambda	정규화 계수
σ	sigma	표준편차, 활성화 함수
∇	nabla	그래디언트 연산자

수학 기호

기호	의미	예시
$\mathbb{E}[\cdot]$	기댓값 (평균)	$\mathbb{E}[X] = 3.5$
$\sum$	합 (시그마)	$\sum_{i=1}^{n} x_i$
$\prod$	곱 (프로덕트)	$\prod_{i=1}^{n} x_i$
$\in$	속한다	$x \in \mathbb{R}$
$\subset$	부분집합	$A \subset B$
$\sim$	~에서 추출	$(x, y) \sim P$
$\approx$	근사	$\pi \approx 3.14$
$\rightarrow$	수렴	$x \rightarrow 0$
$\mathcal{H}$	가설 공간	$h \in \mathcal{H}$
$\mathbb{R}$	실수 집합	$x \in \mathbb{R}$

첨자 표기

표기	의미	예시
x_i	아래 첨자	i번째 데이터

표기	의미	예시
$x^{(t)}$	위 첨자 (괄호)	t번째 시간 단계
x^2	위 첨자 (숫자)	x의 제곱
$\hat{y}$	햇 (hat)	예측값
$\bar{x}$	바 (bar)	평균

연산자

기호	의미	예시
$\frac{\partial}{\partial x}$	편미분	$\frac{\partial L}{\partial \theta}$
$\|\|x\|\|$	노름 (크기)	$\|\|\theta\|\|_2$
$\log$	자연로그	$\log(x)$
$\exp$	지수함수	$\exp(x) = e^x$
$\arg\min$	최소값을 주는 인자	$\arg\min_\theta L(\theta)$