# 2.1 이론

## 2.1.1 분류 문제란?

분류(Classification)는 입력 데이터 $\boldsymbol{x} \in \mathbb{R}^d$ 가 주어졌을 때, 이를 미리 정의된 클래스 집합 중 하나에 할당하는 문제입니다. 여기서 입력 $\boldsymbol{x}$ 는 $d$ 차원의 실수 벡터이며, 출력 클래스 $y$ 는 이산적인 값입니다.

$$f : \mathbb{R}^d \to \{1, 2, \ldots, K\}$$

분류 문제는 목적에 따라 다음과 같이 나뉩니다:

- **이진 분류**: 클래스가 2개 (예: 스팸/정상)
- **다중 클래스 분류**: 클래스가 3개 이상 중 하나 (예: 손글씨 숫자 인식)
- **다중 레이블 분류**: 여러 클래스가 동시에 정답일 수 있음 (예: 영화 장르)

---

## 2.1.2 대표 분류 알고리즘 비교

### 로지스틱 회귀 (Logistic Regression)

선형 결합 후 시그모이드 함수로 확률을 계산하는 모델:

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x} + b)}}$$

결정 경계는 선형: $\boldsymbol{w}^T \boldsymbol{x} + b = 0$

---

### K-최근접 이웃 (KNN)

새로운 입력이 들어왔을 때 가장 가까운 K개의 이웃을 기준으로 다수결 분류. 학습이 따로 없고 직관적이지만, 고차원에서는 성능이 급격히 떨어질 수 있습니다.

---

### 결정 트리 (Decision Tree)

입력을 조건문으로 분기하며 분류를 수행. 해석이 쉬우나 트리 깊이가 깊어지면 과적합 위험이 있습니다.

---

### 서포트 벡터 머신 (SVM)

두 클래스를 **가장 넓은 마진**으로 나누는 결정 경계를 학습합니다:

$\boldsymbol{w}^T \boldsymbol{x} + b = 0$

커널 기법을 이용하면 비선형 경계도 가능하며 일반화 성능이 뛰어납니다. 단점은 학습 속도와 매개변수 조정의 어려움입니다.

---

## 2.1.3 결정 경계와 해석 가능성

각 모델은 입력 공간을 구분짓는 결정 경계(Decision Boundary)를 학습합니다. 이 경계는 예측 기준이 되며, 모델의 성격을 파악하는 핵심 요소입니다.

> 해석력: 모델이 어떻게 예측을 수행하는지, 어떤 변수가 어떤 경향을 미치는지, 사람이 이해할 수 있는 방식으로 설명 가능한 정도입니다.

| 모델 | 경계 형태 | 해석력 | 특이사항 |
| --- | --- | --- | --- |
| 로지스틱 회귀 | 선형 | 매우 높음 | 가중치로 변수 영향도 해석 가능 |
| 결정 트리 | 계단형 | 높음 | 조건 분기 구조 설명 가능 |
| KNN | 매우 유연 | 낮음 | 해석 어려움 |
| SVM | 선형/비선형 | 낮음 | 고차원 해석 어려움 |

## 2.1.4 고려사항

### 특성 스케일링

거리 기반 모델(KNN, SVM)이나 내적(dot product) 기반 모델(로지스틱 회귀)은 스케일 차이에 민감합니다. 따라서 표준화를 통해 다음과 같이 조정합니다.

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

| 기호 | 의미 | 설명 |
|---|---|---|
| $x_j$ | 원래 특성값 | 예: 키 = 172cm |
| $\mu_j$ | 변수 $j$의 평균 | 해당 열 전체의 평균 |
| $\sigma_j$ | 변수 $j$의 표준편차 | 분산의 제곱근 |
| $x'_j$ | 표준화된 값 | 평균 0, 표준편차 1로 변환된 값 |

---

### 클래스 불균형 대응

정확도(Accuracy)만으로는 불균형 문제를 파악할 수 없습니다.(클래스 불균형으로 왜곡이 될 수 있습니다.)

다음과 같은 대응이 필요합니다.

- 데이터: 오버샘플링(소수 클래스의 데이터를 인위적으로 늘리기), 언더샘플링(다수 클래스 일부 제거)
- 모델: `class_weight='balanced'`
- 평가: 정밀도(Precision), 재현율(Recall), F1 점수 활용

---

### 예측 성능 vs 해석 가능성

| 모델 | 성능 | 해석력 | 적합한 사용 예 |
|---|---|---|---|
| **로지스틱 회귀** | 보통 | 매우 높음 | 영향도 분석 |
| **결정 트리** | 보통 | 높음 | 조건 설명 필요할 때 |
| **SVM** | 높음 | 낮음 | 복잡한 결정 경계 |
| **KNN** | 낮음 | 없음 | 단순한 실험용 |
| **신경망** | 매우 높음 | 매우 낮음 | 설명 불필요, 고성능 예측 필요 |

# 2.2 수식

## 2.2.1 로지스틱 회귀 (Logistic Regression)

로지스틱 회귀는 이진 분류 문제를 풀기 위한 대표적인 지도 학습 모델입니다. 입력 벡터 $\mathbf{x} \in \mathbb{R}^d$ 가 주어졌을 때, 클래스 $y \in \{0, 1\}$ 중 하나에 속할 확률을 예측합니다.

### 2.2.1.1 선형 결합과 시그모이드 함수

입력 벡터 $\mathbf{x}$에 대해 먼저 선형 결합을 계산합니다.

$$z = \mathbf{w}^T\mathbf{x} + b$$

이 값을 시그모이드 함수에 통과시켜 확률로 변환합니다.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \Rightarrow \quad \hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

### 2.2.1.2 확률 분포 기반 모델링

출력 클래스 $y \in \{0, 1\}$는 확률값 $\hat{y}$를 따르는 베르누이 분포(Bernoulli Distribution)로 가정합니다.

> 베르누이 시행: 결과가 두 가지 중 하나로만 나오는 실험이나 시행(동전 던지기)을 말합니다.
> 베르누이 확률 변수: 베르누이 시행의 결과를 실수 0 또는 1로 바꾼 것을 말합니다.
> 베르누이 확률 분포: 베르누이 확률변수의 분포를 베르누이 확률분포 혹은 베루누이 분포라고 합니다.

$$P(y \mid \mathbf{x}) = \hat{y}^y(1 - \hat{y})^{1-y}$$

위 수식은 **y가 1일 확률은 $\hat{y}$, 0일 확률은 $1 - \hat{y}$** 를 **하나의 공식으로 통합** 한 것입니다.

| 상황 | 수식 계산 | 의미 |
| --- | --- | --- |
| $y = 1$ | $\hat{y}^1(1 - \hat{y})^0 = \hat{y}$ | 양성 클래스의 확률 |
| $y = 0$ | $\hat{y}^0(1 - \hat{y})^1 = 1 - \hat{y}$ | 음성 클래스의 확률 |

### 2.2.1.3 우도 함수 정의

훈련 데이터셋 $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$에 대해 전체 우도 함수는 다음과 같습니다.

> 우도(likelihood)란, 현재 주어진 모델 파라미터 $(w, b)$에 대해 우리가 관측한 데이터가 나올 확률이 얼마나 높은지를 나타내는 값입니다.
> 즉, 파라미터를 고정시키고 관측 데이터가 나올 확률을 계산해보고 그 확률을 최대화 시켜보자는 것입니다.
> → 파라미터를 바꾸어서 말입니다.

$$\mathcal{L}(\mathbf{w}, b) = \prod_{i=1}^{n} \left[\hat{y}^{(i)}\right]^{y^{(i)}} \left[1 - \hat{y}^{(i)}\right]^{1-y^{(i)}}$$

$$\hat{y}^{(i)} = \sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \quad \text{where} \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

### 2.2.1.4 로그우도 함수 도출

계산을 용이하게 하기 위해 우도에 로그를 취하여 **로그우도 함수**로 변환합니다.

> 곱셈이 덧셈으로 바뀝니다.
> 미분도 간단해집니다.(곱셈의 미분보다 말입니다)
> 음수를 취해서 최소화 문제로 바꾸어도 상관없습니다.

$$J(\mathbf{w}, b) = -\ell(\mathbf{w}, b) = -\sum_{i=1}^{n} \left[ y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

또는 평균 손실로 표현하는 것도 가능합니다. 평균을 구하기 위해서 전체 데이터 수로 나누어 줍니다.

$$J(\mathbf{w}, b) = -\frac{1}{n} \sum_{i=1}^{n} \left[ y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

#### 2.2.1.5 경사 하강법을 위한 미분

이 손실 함수를 최소화하기 위해, 각 파라미터에 대해 기울기를 계산합니다.

**(1) 가중치 $\mathbf{w}$에 대한 편미분:**

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}$$

**(2) 편향 $b$ 에 대한 편미분:**

$$\frac{\partial J}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})$$

### 2.2.2 서포트 벡터 머신 (Support Vector Machine)

SVM은 데이터를 분류하기 위해 결정 경계(선/면)를 학습합니다.

그 과정에서 **클래스 간 마진을 최대화**함으로써 일반화 성능을 높이는 것이 핵심 아이디어입니다.

#### 2.2.2.1 결정 경계와 선형 분리

SVM은 먼저 선형 결정 경계를 정의합니다. 이 경계는 다음과 같은 형태입니다.

$$\mathbf{w}^T \mathbf{x} + b = 0$$

- $\mathbf{x} \in \mathbb{R}^d$: 입력 벡터(예측하고자 하는 데이터 포인트)
- $\mathbf{w}$: 경계면의 방향을 나타내는 법선 벡터(어떤 면에 수직인 방향을 가리키는 벡터입니다.)
- $b$: 절편

분류는 다음 규칙에 따릅니다:

- $y = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T \mathbf{x} + b < 0 \end{cases}$

#### 2.2.2.2 마진의 정의

마진(margin)이란, 두 클래스 데이터 사이에서 결정 경계로부터 가장 가까운 점까지의 거리입니다.

SVM은 이 마진을 가능한 넓게 설정하려고 합니다. 마진의 수학적 표현은 다음과 같습니다.

$$\text{Margin} = \frac{2}{\|\mathbf{w}\|}$$

여기서$\|\mathbf{w}\|$는 벡터 $\mathbf{w}$ 의 유클리디안 노름입니다.

> $\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2 + \cdots + w_d^2}$

#### 2.2.2.3 하드 마진 최적화 문제

모든 데이터가 완벽하게 선형 분리 가능하다고 가정할 때, SVM은 다음과 같은 최적화 문제를 풉니다.

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

이때, 제약 조건은 다음과 같습니다.

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \quad \forall i$$

이 구조는:

- $\frac{1}{2}\|\mathbf{w}\|^2$: 마진 최대화를 위한 최소화 대상
- 제약 조건: 모든 샘플이 경계에서 마진 이상 떨어지도록 강제

### 2.2.2.4 소프트 마진과 패널티 $C$

현실에서는 데이터가 완벽하게 분리되지 않는 경우가 많습니다. 이럴 때는 슬랙 변수 $\xi_i$를 도입해 **오류를 허용**하는 소프트 마진 모델을 사용합니다. 최적화 문제는 다음과 같이 바뀝니다.

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

제약 조건 역시 다음과 같이 바뀝니다.

$$y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

- $C > 0$: 마진 위반에 대한 패널티 계수(마진을 위반한 샘플에 얼마나 강하게 벌점을 줄지를 정합니다.)
- $\xi_i$: 각 데이터 포인트가 마진을 위반한 정도

| Orange UI 항목 | 수식에서의 변수 | 설명 |
|---|---|---|
| Cost | $C$ | 소프트 마진 SVM에서 **마진 위반에 대한 패널티 계수**입니다. |

### 2.2.2.5 커널 함수 개념

SVM은 선형 분리 기준을 가지지만, 현실의 데이터는 종종 **선형적으로 분리할 수 없습니다.**

이런 경우, 입력 데이터를 **고차원 공간으로 매핑**해 선형 분리가 가능하도록 만드는데, 이 과정을 효율적으로 수행하기 위해 커널 함수(Kernel function)를 사용합니다.

커널 함수는 입력 벡터를 직접 고차원으로 올리지 않고도, 고차원에서의 내적 결과를 계산해주는 함수입니다.

가장 대표적인 커널:

- **RBF 커널 (Radial Basis Function kernel)**:

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$$

| 항목 | 설명 |
|---|---|
| $\|\mathbf{x} - \mathbf{x}'\|^2$ | 두 벡터 사이의 거리 (유클리디안 거리 제곱) |
| $\exp(-\gamma \cdot 거리)$ | 거리가 작으면 1에 가까워지고, 크면 0에 가까워짐 |
| $\gamma$ | 얼마나 민감하게 거리 차이를 반영할지 조절하는 하이퍼파라미터 |

**Polynomial 커널**: 입력 벡터를 고차 다항식 특성 공간으로 확장합니다.

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + r)^d$$

## 2.2.3 KNN (K-Nearest Neighbors)

KNN은 **사전 학습이 없는** 알고리즘으로, 새로운 데이터가 들어올 때마다 **기존 학습 데이터와의 거리**를 계산하여 가장 가까운 K개의 이웃을 찾고, **다수결 혹은 평균**으로 결과를 결정합니다.

분류 문제에서는 **다수결**, 회귀 문제에서는 **평균값**을 기반으로 예측합니다.

## 2.2.3.1 거리 측정 방식 (Distance Metrics)

입력 벡터 두 개를 각각 $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$, $\boldsymbol{x'} = (x'_1, x'_2, \ldots, x'_d)$라고 할 때, 다양한 거리 측정 방법은 다음과 같습니다.

### (1) 유클리디안 거리 (Euclidean Distance)

가장 흔히 쓰이는 "직선 거리"입니다.

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$$

### (2) 맨해튼 거리 (Manhattan Distance)

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{i=1}^{d} |x_i - x'_i|$$

### (3) 체비쇼프 거리 (Chebyshev Distance, Maximal Distance)

가장 큰 차이 하나를 기준으로 계산합니다.

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \max_i |x_i - x'_i|$$

### (4) 마할라노비스 거리 (Mahalanobis Distance)

공분산 행렬 $\boldsymbol{S}$를 고려한 거리로, 데이터의 분포를 반영합니다.

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \sqrt{(\boldsymbol{x} - \boldsymbol{x'})^\top \boldsymbol{S}^{-1} (\boldsymbol{x} - \boldsymbol{x'})}$$

## 2.2.3.2 가중치 방식 (Weights)

K개의 이웃에 대해 결과를 집계할 때 가중치를 줄 수도 있습니다.

- **Uniform**: 모든 이웃에 동일한 가중치 → 단순 다수결
- **Distance-based**: 가까운 이웃에게 더 큰 가중치를 부여

거리 기반 가중치의 예:

$$w_i = \frac{1}{d(\boldsymbol{x}, \boldsymbol{x}_i) + \varepsilon}$$

## 2.2.3.3 예측 수식

### 분류 (Classification)

가장 많이 등장한 클래스를 선택합니다.

$$\hat{y} = \arg\max_{c \in \mathcal{C}} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} w_i \cdot \mathbf{1}[y_i = c]$$

| 기호 | 의미 |
|---|---|
| $\hat{y}$ | 최종 예측 클래스 |
| $\mathcal{C}$ | 가능한 클래스 집합 (예: {0, 1}) |
| $\mathcal{N}_k(\boldsymbol{x})$ | 입력 $\boldsymbol{x}$의 주변 K개의 최근접 이웃의 인덱스 집합 |
| $y_i$ | 이웃 $i$의 실제 클래스 레이블 |
| $w_i$ | 이웃 $i$의 가중치 (예: 거리의 역수 등) |
| $\mathbf{1}[y_i = c]$ | $y_i$가 클래스 $c$일 경우 1, 아니면 0인 **Indicator Function** |

| 이웃 번호 | 레이블 $y_i$ | 거리 $d_i$ | 가중치 $w_i = \frac{1}{d_i}$ |
|---|---|---|---|
| 1 | 0 | 0.5 | 2.0 |
| 2 | 1 | 0.4 | 2.5 |
| 3 | 1 | 0.3 | 3.33 |

| 이웃 번호 | 레이블 $y_i$ | 거리 $d_i$ | 가중치 $w_i = \frac{1}{d_i}$ |
|---|---|---|---|
| 4 | 0 | 0.6 | 1.67 |
| 5 | 1 | 0.2 | 5.0 |

**클래스 0:**

- 이웃 1: $y_1 = 0, \quad w_1 = 2.0$
- 이웃 4: $y_4 = 0, \quad w_4 = 1.67$

**합계: 3.67**

**클래스 1:**

- 이웃 2: $y_2 = 1, \quad w_2 = 2.5$
- 이웃 3: $y_3 = 1, \quad w_3 = 3.33$
- 이웃 5: $y_5 = 1, \quad w_5 = 5.0$

**합계: 10.83**

## 2.2.4 결정 트리 (Decision Tree)

결정 트리는 입력 특성에 대해 **조건 분기를 반복**하여 데이터를 분류하는 트리 기반 모델입니다.

루트 노드부터 시작해 **특정 속성의 조건**에 따라 데이터를 분할하며,

각 리프 노드는 최종 분류 결과 또는 예측값을 나타냅니다.

- 분류(Classification) 문제에서는 클래스 레이블이 리프 노드에 배정됩니다.
- 회귀(Regression) 문제에서는 리프 노드에 평균값이 배정됩니다

---

### 2.2.4.2 노드 분할 기준: 불순도 (Impurity)

결정 트리는 각 노드에서 불순도(impurity)가 가장 많이 줄어드는 방향으로 데이터를 분할합니다.

**(1) 지니 불순도 (Gini Impurity)**

$$G(t) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

여기서 $p_k$는 노드 $t$에서 클래스 $k$가 등장할 확률입니다.

| 클래스 | 샘플 수 | 비율 $p_k$ |
|---|---|---|
| 0 | 4 | $\frac{4}{10} = 0.4$ |
| 1 | 6 | $\frac{6}{10} = 0.6$ |

$$G(t) = \sum_{k=1}^{K} p_k(1 - p_k) \quad = 0.4(1 - 0.4) + 0.6(1 - 0.6) \quad = 0.4 \cdot 0.6 + 0.6 \cdot 0.4 \quad = 0.24 + 0.24 \quad = 0.48$$

$$G(t) = 1 - \sum_{k=1}^{K} p_k^2 \quad = 1 - (0.4^2 + 0.6^2) \quad = 1 - (0.16 + 0.36) \quad = 1 - 0.52 \quad = 0.48$$

**(2) 엔트로피 (Entropy)**

$$H(t) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

정보 이론에서 유래된 개념으로, 분포의 불확실성을 측정합니다.

### 2.2.4.3 정보 이득 (Information Gain)

정보 이득은 **부모 노드의 불순도**에서 **자식 노드들의 가중 평균 불순도**를 뺀 값입니다.

즉, 이 분할로 얼마나 정돈됐는가를 수치화합니다.

$$IG = I(\text{부모}) - \left( \frac{N_L}{N} I(L) + \frac{N_R}{N} I(R) \right)$$

- $I(\cdot)$: 지니 또는 엔트로피
- $N_L, N_R$: 좌/우 자식 노드의 샘플 수
- $N$: 전체 샘플 수
- 정보 이득이 가장 높은 속성과 임계값을 찾아 분할(split)을 수행합니다.

example:

| 샘플 | 나이 | 클래스 |
|---|---|---|
| A | 22 | 0 |
| B | 25 | 0 |
| C | 28 | 1 |
| D | 30 | 0 |
| E | 32 | 1 |
| F | 35 | 1 |
| G | 38 | 1 |
| H | 40 | 1 |
| I | 42 | 1 |
| J | 45 | 0 |

$$p_0 = \frac{4}{10} = 0.4$$
$$p_1 = \frac{6}{10} = 0.6$$

전체 불순도

$$G(t) = 1 - \left(p_0^2 + p_1^2\right) \quad = 1 - (0.4^2 + 0.6^2) \quad = 1 - (0.16 + 0.36) \quad = 1 - 0.52 \quad = 0.48$$

분할 시도

나이 < 30

왼쪽 그룹 (나이 < 30): A, B, C → 클래스 0, 0, 1

→ 클래스 비율은

$$p_0 = \frac{2}{3}, \quad p_1 = \frac{1}{3}$$

$$G_L = 1 - \left( \left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2 \right) = 1 - \left( \frac{4}{9} + \frac{1}{9} \right) = 1 - \frac{5}{9} = \frac{4}{9} \approx 0.444$$

오른쪽 그룹 (나이 ≥ 30): D ~ J → 클래스 0, 1, 1, 1, 1, 1, 0

→ 클래스 비율은

$$p_0 = \frac{2}{7}, \quad p_1 = \frac{5}{7}$$

$$G_R = 1 - \left( \left(\frac{2}{7}\right)^2 + \left(\frac{5}{7}\right)^2 \right) = 1 - \left( \frac{4}{49} + \frac{25}{49} \right) = 1 - \frac{29}{49} = \frac{20}{49} \approx 0.408$$

## 2.2.4.5 과적합과 가지치기 (Overfitting & Pruning)

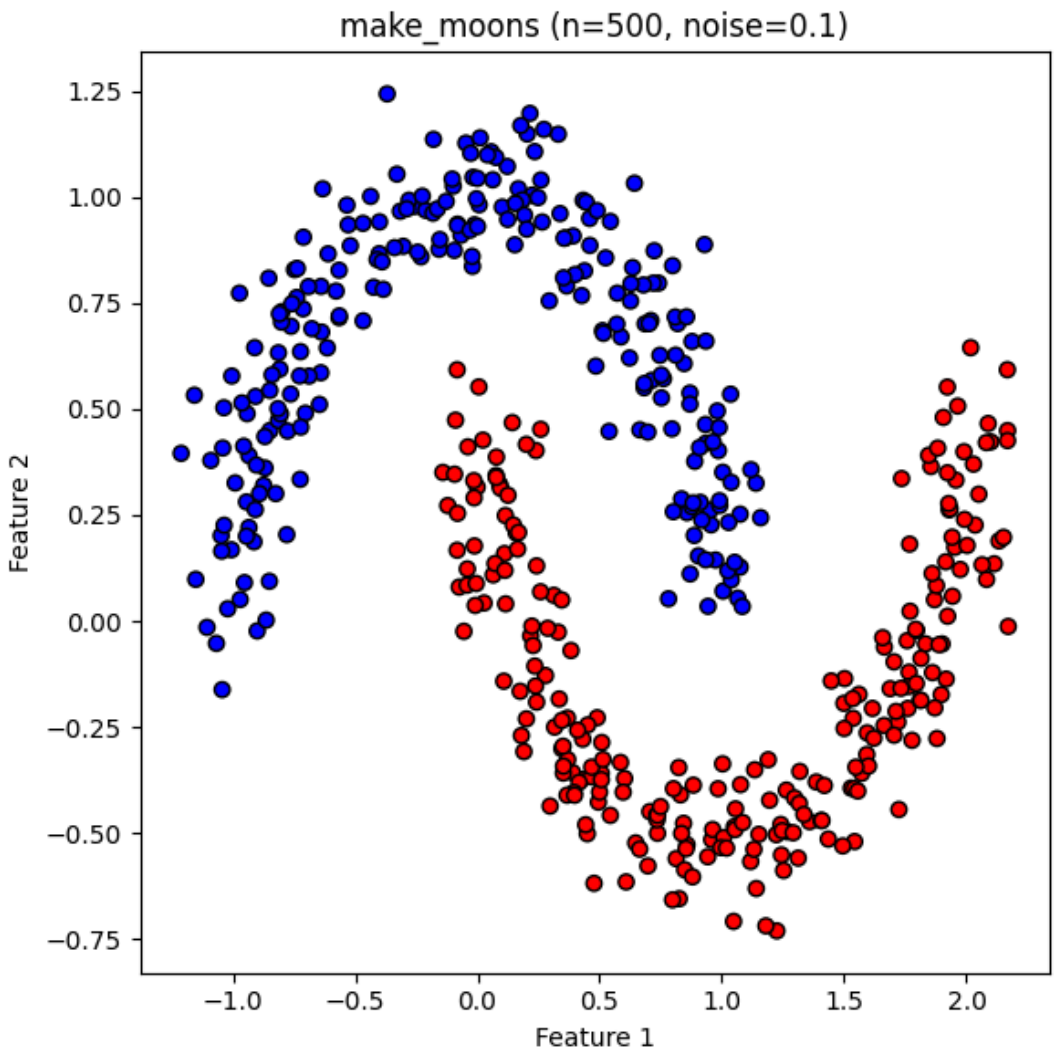
- 결정 트리는 과적합에 취약합니다. 모든 노드를 다 분할하면 훈련 데이터에 너무 잘 맞지만 일반화 성능이 낮아질 수 있습니다.

- 이를 방지하기 위해 다음과 같은 기법을 사용합니다.

| | 설명 |
|---|---|
| 사전 가지치기 (Pre-pruning) | 트리 깊이 제한, 최소 샘플 수 제한 등 조건 미리 설정 |
| 사후 가지치기 (Post-pruning) | 완성된 트리에서 덜 중요한 노드를 제거 |

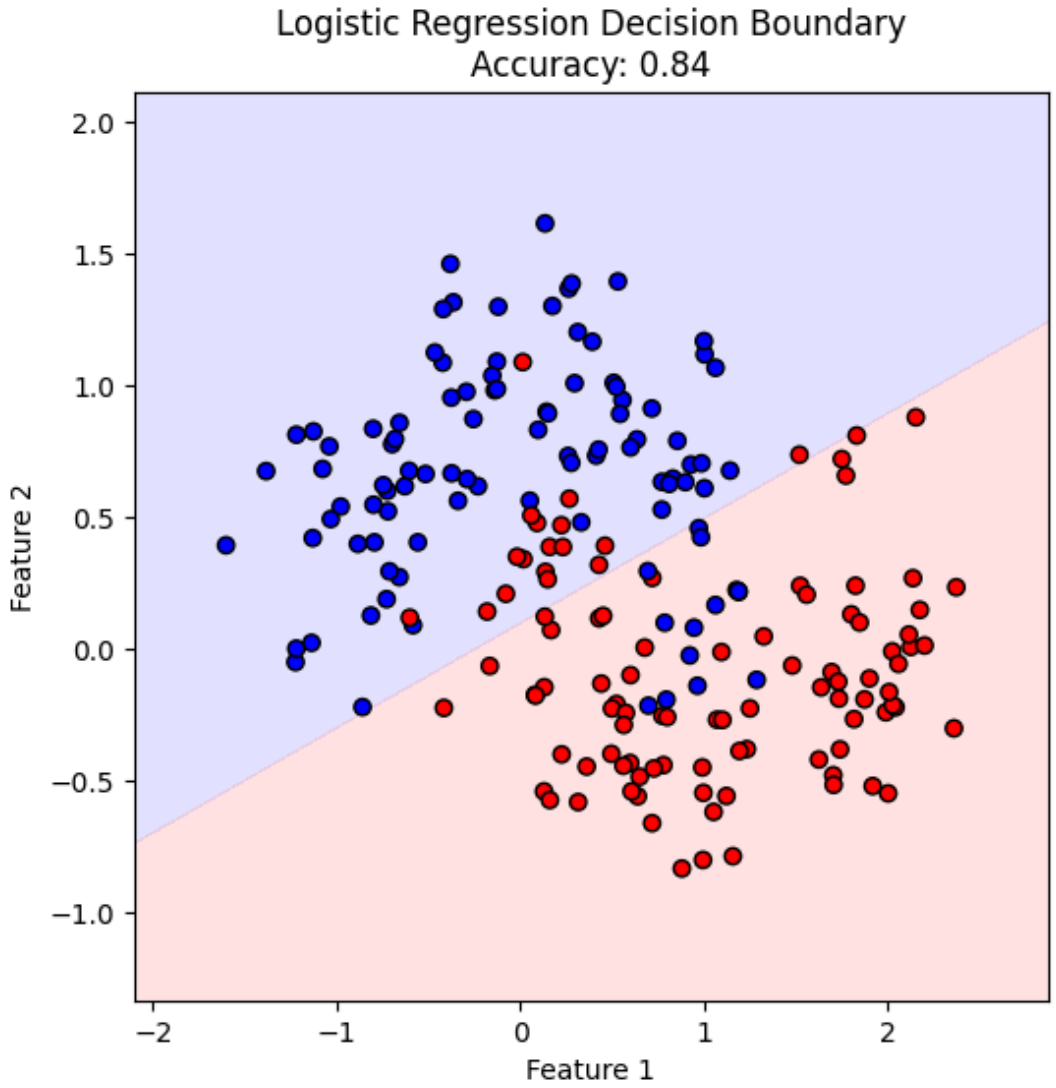
# 2.3 코드

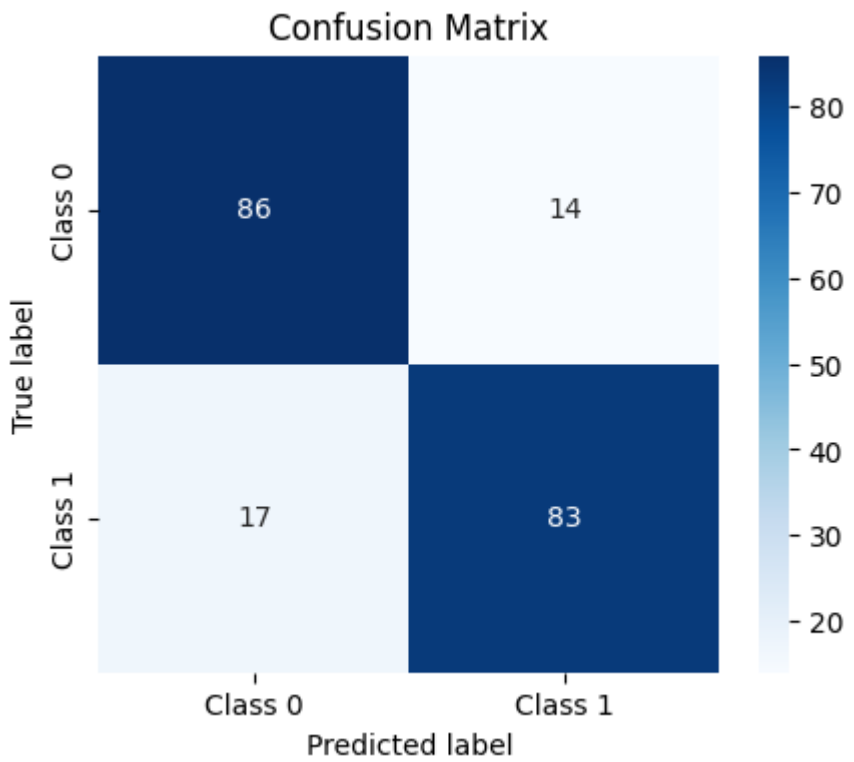
```
#1. 라이브러리 임포트
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

#2. 데이터 생성 및 시각화
# 샘플 수: 500, 노이즈: 0.1, 랜덤 시드: 42
dataset = make_moons(n_samples=500, noise=0.1, random_state=42)
X, y = dataset

plt.figure(figsize=(6,6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', edgecolor='k')
plt.title('make_moons (n=500, noise=0.1)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.tight_layout()
plt.show()
```




```
#3. 로지스틱 회귀 – 결정 경계 · Seaborn Heatmap 혼동 행렬 · Classification Report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# 모델 학습
model_lr = LogisticRegression()
```

```
model_lr.fit(X, y)
y_pred = model_lr.predict(X)
acc = accuracy_score(y, y_pred)

# ── 결정 경계 시각화 ──
xx, yy = np.meshgrid(
    np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 200),
    np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 200)
)
grid = np.c_[xx.ravel(), yy.ravel()]
probs = model_lr.predict_proba(grid)[:,1].reshape(xx.shape)

plt.figure(figsize=(6,6))
plt.contourf(xx, yy, probs, levels=[0, 0.5, 1], alpha=0.2, cmap='bwr')
plt.scatter(X[:,0], X[:,1], c=y, edgecolor='k', cmap='bwr')
plt.title(f'Logistic Regression Decision Boundary\nAccuracy: {acc:.2f}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# ── Seaborn Heatmap으로 혼동 행렬 ──
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0','Class 1'],
            yticklabels=['Class 0','Class 1'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# ── Classification Report ──
print(f'Accuracy: {acc:.2f}\n')
print(classification_report(y, y_pred))
```



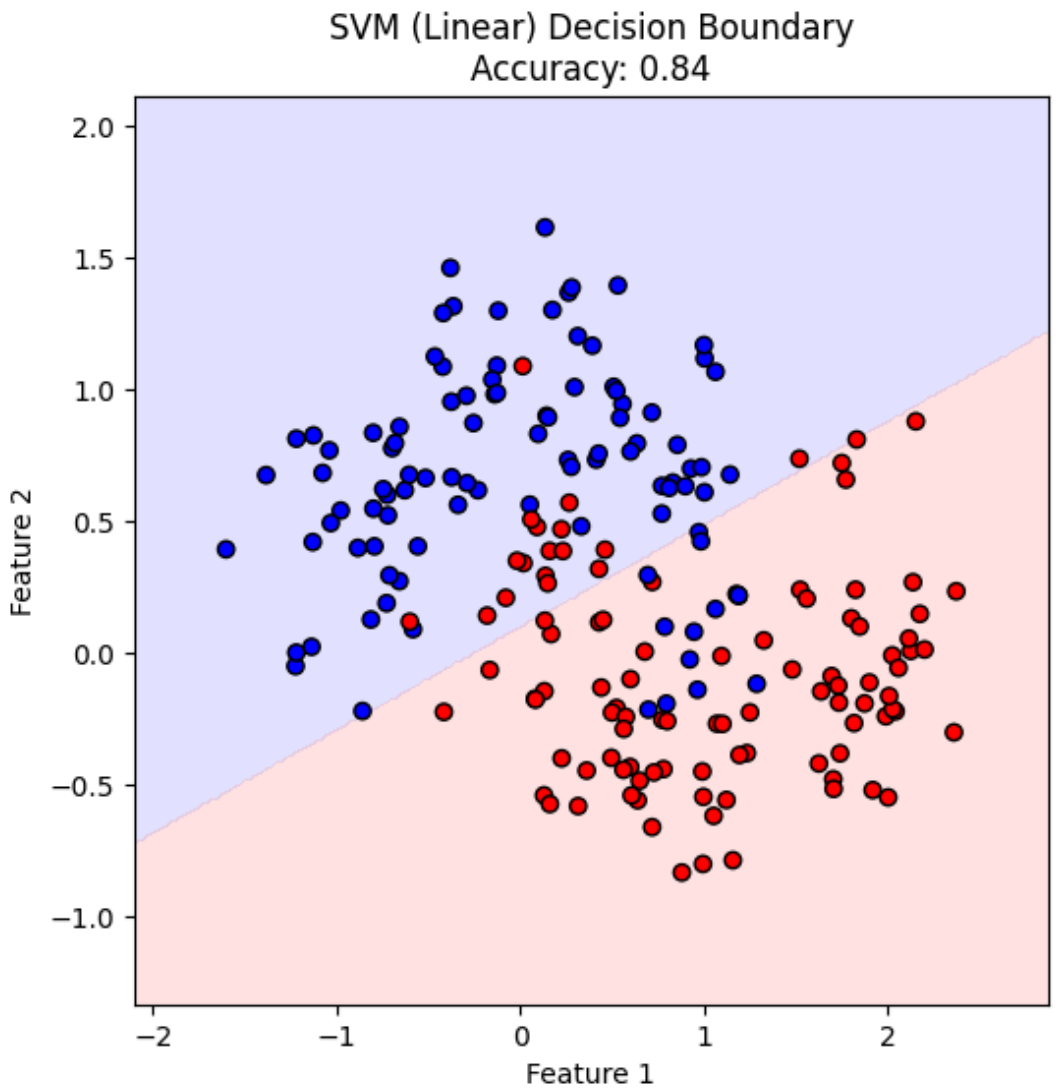

Accuracy: 0.84

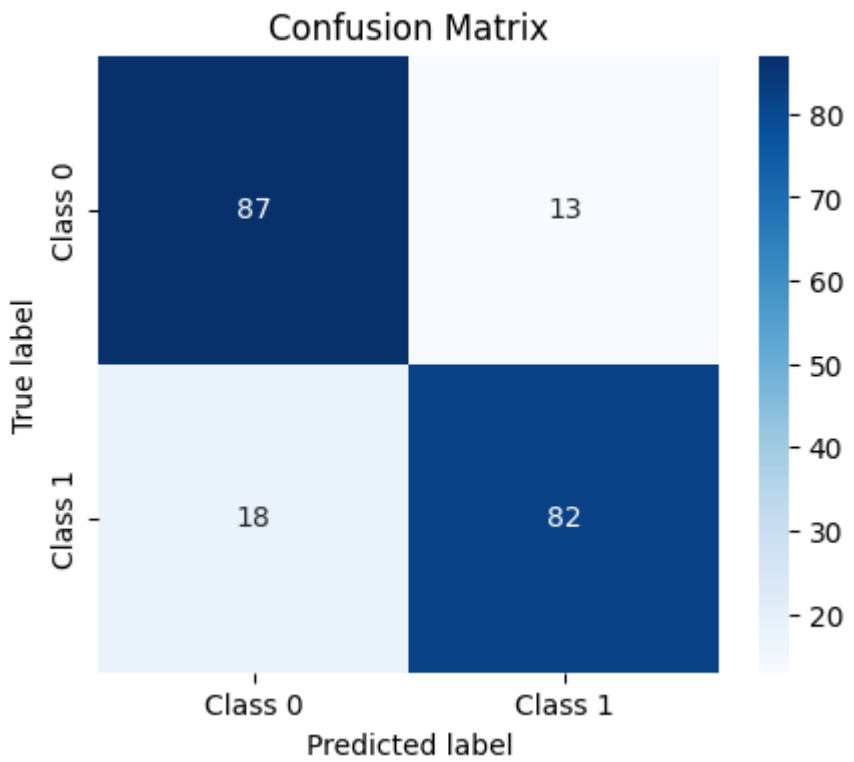
```
              precision    recall  f1-score   support

           0       0.83      0.86      0.85       100
           1       0.86      0.83      0.84       100

    accuracy                           0.84       200
```

macro avg 0.85 0.84 0.84 200
weighted avg 0.85 0.84 0.84 200

```
# SVM (Linear Kernel) – 결정 경계 · 혼동 행렬 · Classification Report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.svm import SVC           # 또는: from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# ── 모델 학습 ──
model_svm_lin = SVC(kernel='linear', probability=True, random_state=42)
# 만약 LinearSVC를 쓰고 싶다면:
# from sklearn.svm import LinearSVC
# model_svm_lin = LinearSVC(random_state=42)
model_svm_lin.fit(X, y)
y_pred = model_svm_lin.predict(X)
acc = accuracy_score(y, y_pred)

# ── 결정 경계 시각화 ──
xx, yy = np.meshgrid(
    np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 200),
    np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 200)
)
grid = np.c_[xx.ravel(), yy.ravel()]
probs = model_svm_lin.predict_proba(grid)[:,1].reshape(xx.shape)

plt.figure(figsize=(6,6))
plt.contourf(xx, yy, probs, levels=[0, 0.5, 1], alpha=0.2, cmap='bwr')
plt.scatter(X[:,0], X[:,1], c=y, edgecolor='k', cmap='bwr')
plt.title(f'SVM (Linear) Decision Boundary\nAccuracy: {acc:.2f}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# ── Seaborn Heatmap으로 혼동 행렬 ──
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0','Class 1'],
            yticklabels=['Class 0','Class 1'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# ── Classification Report ──
print(f'Accuracy: {acc:.2f}\n')
print(classification_report(y, y_pred))
```



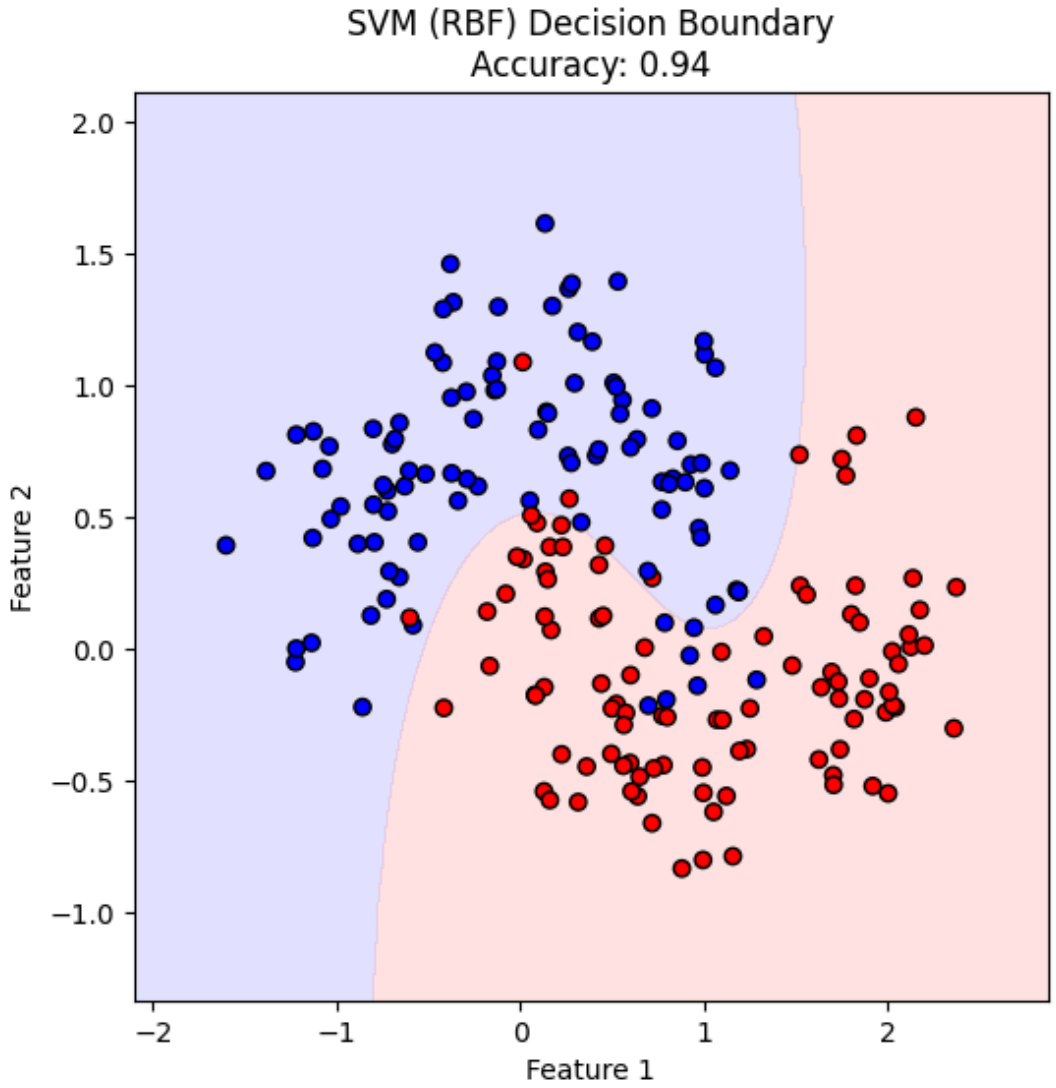

Accuracy: 0.84

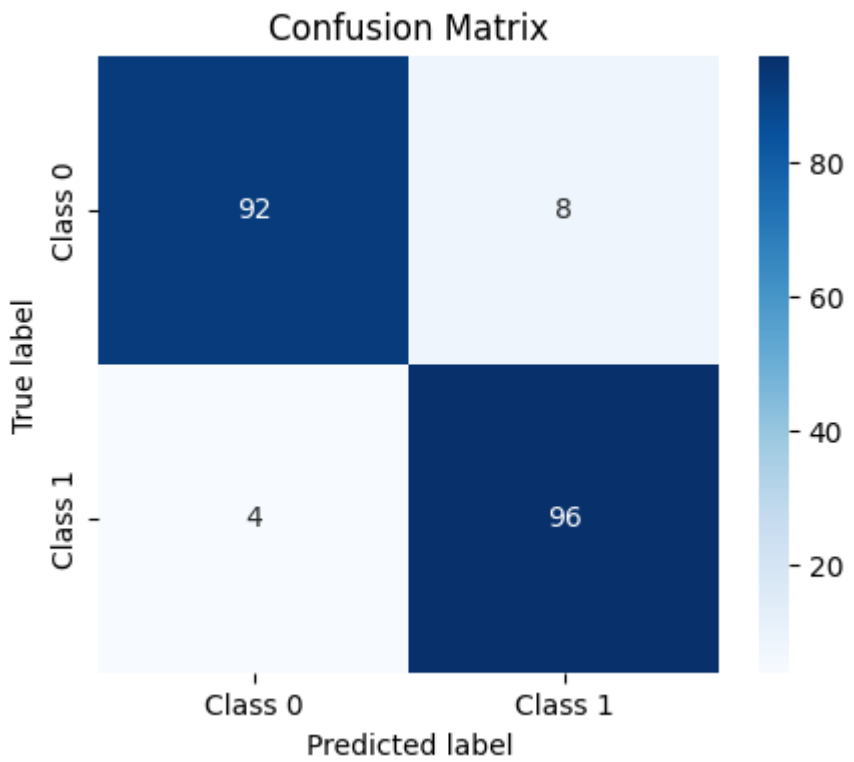
```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       100
           1       0.86      0.82      0.84       100

    accuracy                           0.84       200
```

macro avg 0.85 0.84 0.84 200
weighted avg 0.85 0.84 0.84 200

```
#4. SVM (RBF 커널) – 결정 경계 · Seaborn Heatmap 혼동 행렬 · Classification Report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.svm import SVC #support vector classifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# ── 모델 학습 ──
model_svm = SVC(kernel='rbf', probability=True, gamma='scale')
model_svm.fit(X, y)
y_pred = model_svm.predict(X)
acc = accuracy_score(y, y_pred)

# ── 결정 경계 시각화 ──
xx, yy = np.meshgrid(
    np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 200),
    np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 200)
)
grid = np.c_[xx.ravel(), yy.ravel()]
probs = model_svm.predict_proba(grid)[:,1].reshape(xx.shape)

plt.figure(figsize=(6,6))
plt.contourf(xx, yy, probs, levels=[0, 0.5, 1], alpha=0.2, cmap='bwr')
plt.scatter(X[:,0], X[:,1], c=y, edgecolor='k', cmap='bwr')
plt.title(f'SVM (RBF) Decision Boundary\nAccuracy: {acc:.2f}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# ── Seaborn Heatmap으로 혼동 행렬 ──
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0','Class 1'],
            yticklabels=['Class 0','Class 1'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# ── Classification Report ──
print(f'Accuracy: {acc:.2f}\n')
print(classification_report(y, y_pred))
```



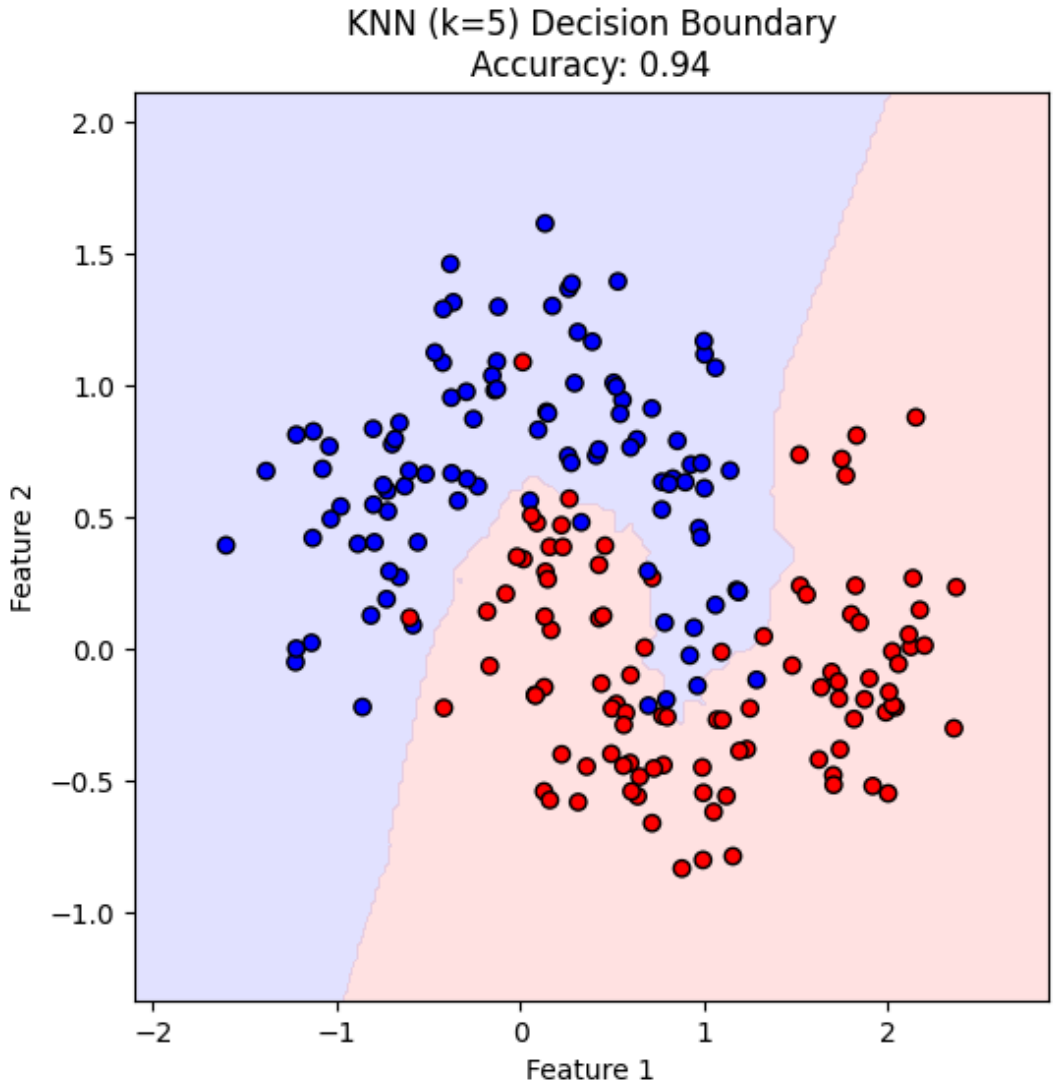

Accuracy: 0.94

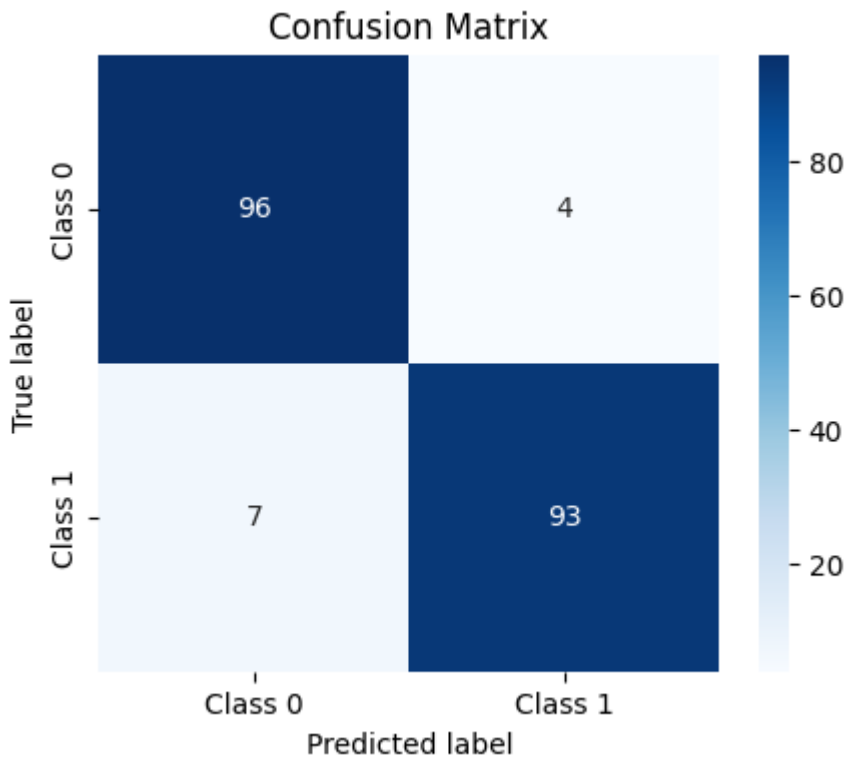
```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       100
           1       0.92      0.96      0.94       100

    accuracy                           0.94       200
```

macro avg 0.94 0.94 0.94 200
weighted avg 0.94 0.94 0.94 200

```
#5. KNN (k=5) – 결정 경계 · Seaborn Heatmap 혼동 행렬 · Classification Report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# ── 모델 학습 ──
model_knn = KNeighborsClassifier(n_neighbors=5)
model_knn.fit(X, y)
y_pred = model_knn.predict(X)
acc = accuracy_score(y, y_pred)

# ── 결정 경계 시각화 ──
xx, yy = np.meshgrid(
    np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 200),
    np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 200)
)
grid = np.c_[xx.ravel(), yy.ravel()]
probs = model_knn.predict_proba(grid)[:,1].reshape(xx.shape)

plt.figure(figsize=(6,6))
plt.contourf(xx, yy, probs, levels=[0, 0.5, 1], alpha=0.2, cmap='bwr')
plt.scatter(X[:,0], X[:,1], c=y, edgecolor='k', cmap='bwr')
plt.title(f'KNN (k=5) Decision Boundary\nAccuracy: {acc:.2f}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# ── Seaborn Heatmap으로 혼동 행렬 ──
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0','Class 1'],
            yticklabels=['Class 0','Class 1'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# ── Classification Report ──
print(f'Accuracy: {acc:.2f}\n')
print(classification_report(y, y_pred))
```



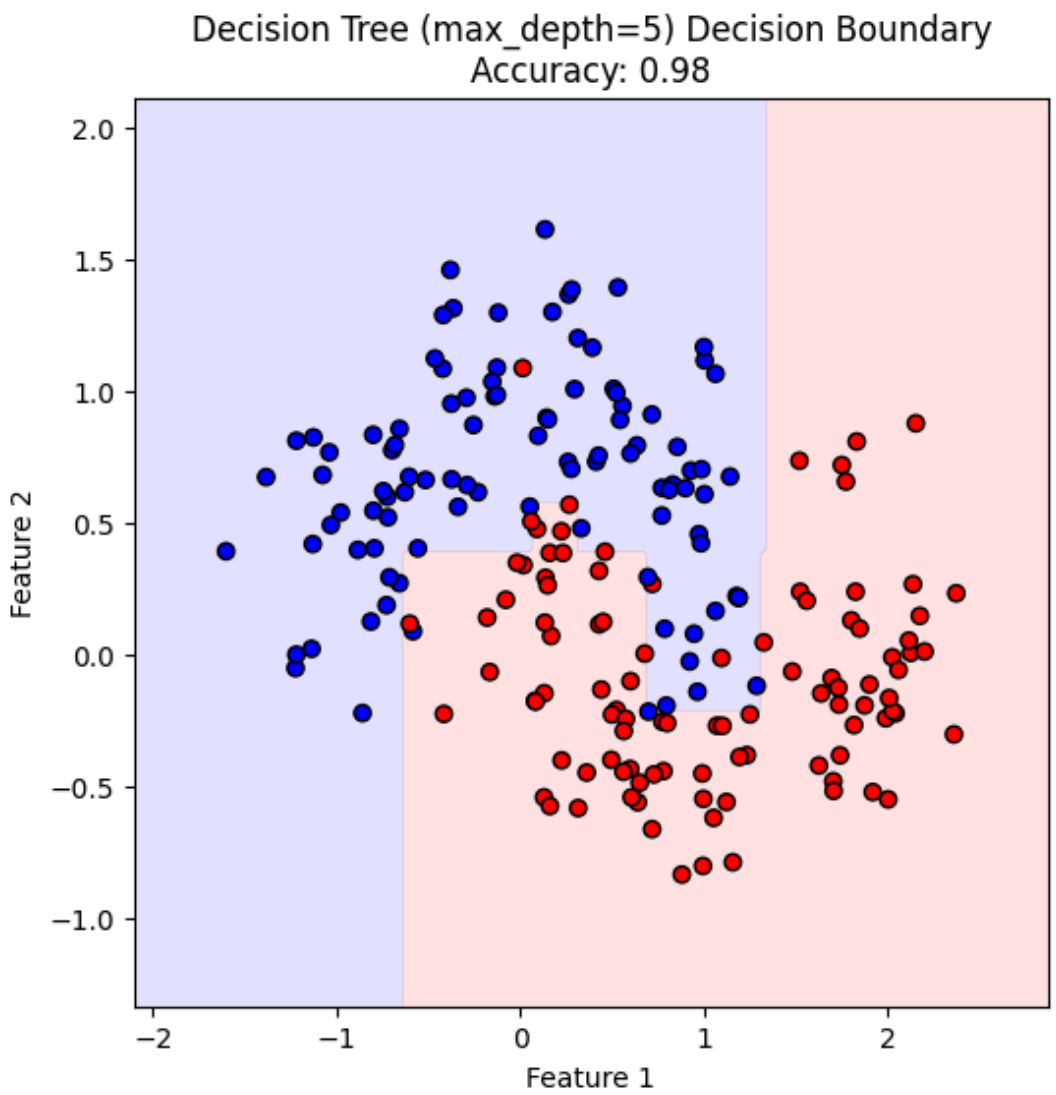

Accuracy: 0.94

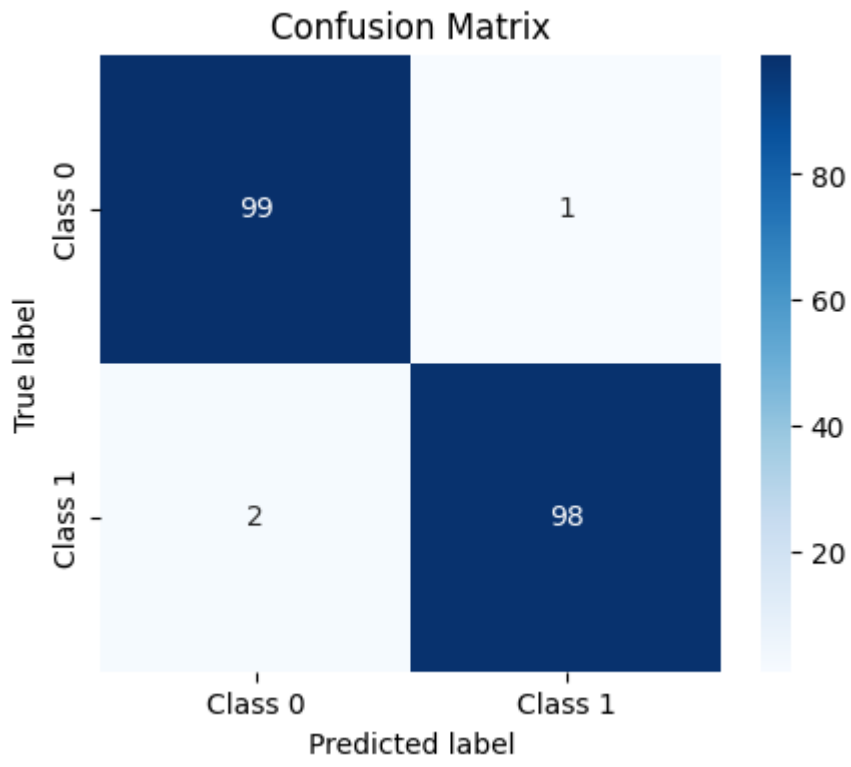
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.93 | 0.96 | 0.95 | 100 |
| 1 | 0.96 | 0.93 | 0.94 | 100 |
| accuracy | | | 0.94 | 200 |
| macro avg | 0.95 | 0.95 | 0.94 | 200 |
| weighted avg | 0.95 | 0.94 | 0.94 | 200 |

#6. Decision Tree (max_depth=5) – 결정 경계 · Seaborn Heatmap 혼동 행렬 · Classification Report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# ── 모델 학습 ──
model_dt = DecisionTreeClassifier(max_depth=5, random_state=42)
model_dt.fit(X, y)
y_pred = model_dt.predict(X)
acc = accuracy_score(y, y_pred)

# ── 결정 경계 시각화 ──
xx, yy = np.meshgrid(
    np.linspace(X[:,0].min()-0.5, X[:,0].max()+0.5, 200),
    np.linspace(X[:,1].min()-0.5, X[:,1].max()+0.5, 200)
)
grid = np.c_[xx.ravel(), yy.ravel()]
probs = model_dt.predict_proba(grid)[:,1].reshape(xx.shape)

plt.figure(figsize=(6,6))
plt.contourf(xx, yy, probs, levels=[0, 0.5, 1], alpha=0.2, cmap='bwr')
plt.scatter(X[:,0], X[:,1], c=y, edgecolor='k', cmap='bwr')
plt.title(f'Decision Tree (max_depth=5) Decision Boundary\nAccuracy: {acc:.2f}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# ── Seaborn Heatmap으로 혼동 행렬 ──
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0','Class 1'],
            yticklabels=['Class 0','Class 1'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# ── Classification Report ──
print(f'Accuracy: {acc:.2f}\n')
print(classification_report(y, y_pred))


#7. Decision Tree 구조 시각화
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 8))
plot_tree(
    model_dt,
    max_depth=2,
    feature_names=['Feature 1', 'Feature 2'],
    class_names=['Class 0', 'Class 1'],
    filled=True,
    rounded=True,
    fontsize=10
)
plt.title('Decision Tree Structure')
```

```
plt.show()
```




Accuracy: 0.98

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       100
           1       0.99      0.98      0.98       100

    accuracy                           0.98       200
```

macro avg 0.99 0.98 0.98 200
weighted avg 0.99 0.98 0.98 200

Decision Tree Structure
Feature 2 <= 0.391
gini = 0.5
samples = 200
value = [100, 100]
class = Class 0
True
False
Feature 1 <= -0.629
gini = 0.307
samples = 111
value = [21, 90]
class = Class 1
Feature 1 <= 1.333
gini = 0.199
samples = 89
value = [79, 10]
class = Class 0
gini = 0.0
samples = 8
value = [8, 0]
class = Class 0
Feature 2 <= -0.22
gini = 0.221
samples = 103
value = [13, 90]
class = Class 1
Feature 2 <= 0.583
gini = 0.112
samples = 84
value = [79, 5]
class = Class 0
gini = 0.0
samples = 5
value = [0, 5]
class = Class 1
(...)
(...)
(...)
(...)

# 2.4 결과 해석

## 2.4.1 오차 행렬 (Confusion Matrix)

| | 예측 양성 (Pred +) | 예측 음성 (Pred –) |
|---|---|---|
| 실제 양성 (True +) | **TP(True Positive)** | **FN(False Negative)** |
| 실제 음성 (True –) | **FP(False Positive)** | **TN(True Negative)** |

- **TP (True Positive)**: 실제 양성을 양성으로 맞춘 경우
- **TN (True Negative)**: 실제 음성을 음성으로 맞춘 경우
- **FP (False Positive)**: 실제 음성을 양성으로 잘못 예측
- **FN (False Negative)**: 실제 양성을 음성으로 잘못 예측

## 2.4.2 정확도 (Accuracy)

모델이 전체 데이터 중에서 얼마나 많이 정답을 맞췄는지를 나타내는 지표입니다.

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

이해하기 쉬워 널리 사용되지만, 클래스 불균형 상황에서는 신뢰하기 어려운 단점이 있습니다.

## 2.4.3 정밀도·재현율·F1 스코어 (Precision, Recall, F1-score)

**정밀도(Precision): 양성으로 예측한 것 중 실제 양성의 비율**

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

- 거짓 양성(FP)을 얼마나 줄였는지를 보여줍니다.
- 예: 이메일 스팸 필터에서 정상 메일을 스팸으로 잘못 분류하면 문제가 크므로 정밀도가 중요합니다.

**재현율(Recall) 또는 민감도(Sensitivity): 실제 양성 중 양성으로 맞춘 비율**

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

- 거짓 음성(FN)을 얼마나 줄였는지를 보여줍니다.
- 예: 암 진단 모델에서는 암 환자를 놓치지 않는 것이 중요하므로 재현율이 중요합니다.

**F1 스코어: 정밀도와 재현율의 조화평균**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

> 조화평균: $H = \frac{2ab}{a+b}$
> 두 값 중 작은 값에 더 민감, 두 값이 모두 클 때만  조화 평균이 크게 나옴.

- 정밀도와 재현율이 모두 중요한 경우 F1-score가 유용합니다.
- **불균형 데이터셋**에서 특히 많이 사용됩니다.

## 2.4.4 ROC 커브와 AUC (Receiver Operating Characteristic & Area Under Curve)

ROC 곡선은 모델의 임계값(threshold)을 변화시킬 때의 민감도(Recall)와 **특이도의 보완 지표(FPR)** 사이의 관계를 시각화한 것입니다.

| 항목 | 정의 | 수식 |
|---|---|---|
| **민감도** (Sensitivity, Recall) | 실제 양성 중 양성으로 맞춘 비율 | $\frac{TP}{TP+FN}$ |
| **특이도** (Specificity) | 실제 음성 중 음성으로 맞춘 비율 | $\frac{TN}{TN+FP}$ |

| 항목 | 정의 | 수식 |
| --- | --- | --- |
| FPR (False Positive Rate) | 잘못 양성으로 예측한 비율 | $\frac{FP}{TN+FP} = 1 - \text{Specificity}$ |

### 2.4.4.1 ROC 커브란?

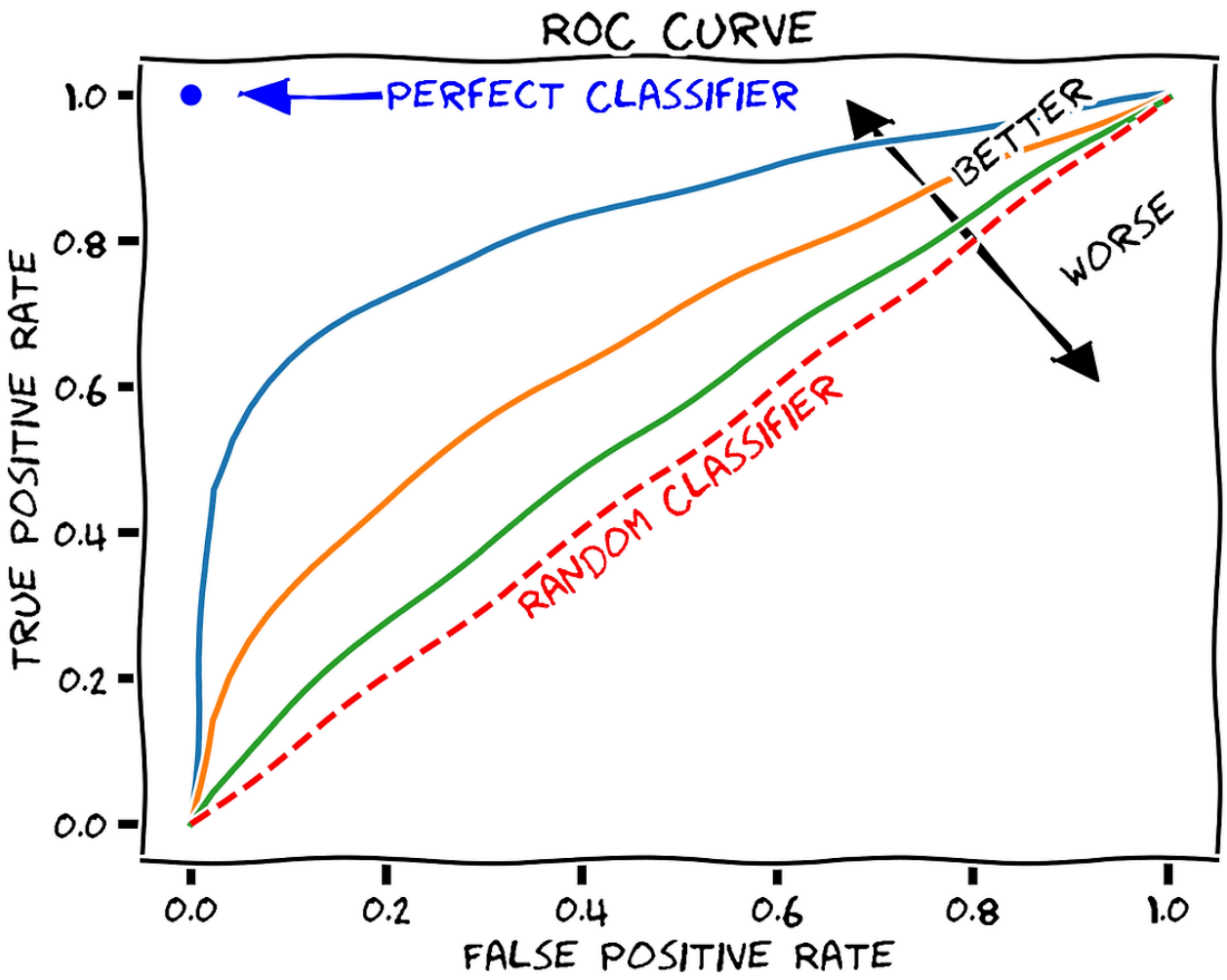
**ROC (Receiver Operating Characteristic) 커브**는 분류 모델의 성능을 **임계값(threshold)** 변화에 따라 시각화한 곡선입니다.

- x축: **FPR (False Positive Rate)** → 잘못 양성으로 예측한 비율
- y축: **TPR (True Positive Rate)** → 실제 양성 중에 양성으로 맞춘 비율 (Recall)

**사용 이유:**

이진 분류 모델은 일반적으로 확률을 출력하고, 그 값을 기준 임계값(예: 0.5)으로 양성/음성을 나눕니다.

- 임계값을 **0.0 ~ 1.0 사이로 조금씩 변경**해가며, 각 경우마다 TPR과 FPR을 계산합니다.
- 이 **(FPR, TPR)** 좌표를 잇는 선이 ROC 커브입니다.
- 좋은 모델은 ROC 커브가 **왼쪽 위 모서리에 가까울수록** 좋습니다 (높은 TPR, 낮은 FPR)
-




https://en.wikipedia.org/

### 2.4.4.2 AUC(Area Under the Curve)

- ROC 커브 아래의 **면적 (Area Under the Curve)**
- 값의 범위: $0.5 \leq \text{AUC} \leq 1.0$
  - AUC = 1: 완벽한 분류기
  - AUC = 0.5: 무작위 분류기 수준
  - AUC < 0.5: 잘못된 방향으로 분류 (거꾸로 뒤집으면 더 좋음)