

교차검증과 모델 평가

올바르게 평가해야 올바르게 개선할 수 있다

이 가이드는 ML 모델을 만들었지만 성능을 어떻게 믿을 수 있는지 막막한
입문자를 대상으로 합니다. 회귀와 분류 모두 다룹니다.

1. 들어가며

모델을 만들었다면 가장 먼저 드는 질문은 “이 모델을 믿어도 되는가?”입니다. 그런데 훈련 데이터의 성능만 보고 이 질문에 답하면 틀릴 가능성이 높습니다. 모델이 훈련 데이터에서 좋은 성능을 낸다고 해서 새로운 데이터에서도 좋은 성능을 낸다는 보장이 없기 때문입니다.

이 가이드는 두 가지 핵심 질문에 답합니다. 첫째, 모델의 성능을 어떻게 측정해야 신뢰할 수 있는가(교차검증). 둘째, 어떤 숫자로 성능을 표현해야 하는가(평가지표).

예제 데이터셋

이 가이드는 두 가지 데이터셋을 병행해서 사용합니다.

- 회귀 예제: Diabetes 데이터셋(규제 선형회귀 가이드와 동일) — 당뇨병 환자 442 명의 1 년 후 질병 진행도 예측
- 분류 예제: Breast Cancer Wisconsin 데이터셋(sklearn 내장) — 유방암 조직 검사 결과로 악성/양성 종양 분류 (569 명, 30 개 변수). sklearn 원본은 양성 종양(benign)이 1 로 코딩되어 있지만, 이 시리즈에서는 관심 사건인 악성(malignant)을 1(positive 클래스)로 뒤집어 사용합니다($y = 1 - \text{target}$). 이하 “양성 종양”은 benign 을, “positive 클래스”는 모델이 1 로 예측하는 클래스(=악성)를 뜻합니다.

분류 예제로 Breast Cancer 를 선택한 이유는 “암 진단”이라는 맥락이 “왜 Accuracy 보다 Recall 이 중요한가”를 직관적으로 설명해주기 때문입니다 — 암을 놓치는 것(FN)이 정상을 암으로 잘못 분류하는 것(FP)보다 훨씬 위험하다는 점이 누구에게나 바로 납득됩니다.

2. 데이터 분할의 원칙

2-1. 훈련 / 검증 / 테스트의 역할 구분

모델 평가를 제대로 하려면 데이터를 목적에 따라 엄격히 분리해야 합니다.

- 훈련 데이터: 모델의 계수(파라미터)를 학습시키는 데 사용합니다.
- 검증 데이터: 하이퍼파라미터(예: λ , 트리 깊이)를 조정하는 데 사용합니다. 교차검증의 각 fold 가 이 역할을 합니다.
- 테스트 데이터: 최종 모델의 성능을 딱 한 번만 측정하는 데 사용합니다. 모델 개발 과정에서 절대 건드리지 않습니다.

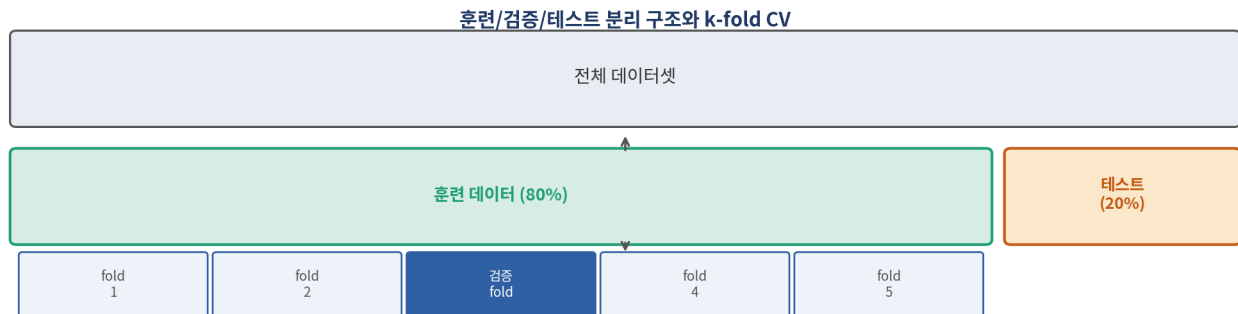


그림 1. 훈련/검증/테스트 분리 구조와 k-fold CV 의 관계. 훈련 데이터를 5 개 fold 로 나눠, 한 번에 하나씩 검증용으로 사용하며 나머지로 학습한다.

테스트 데이터를 “딱 한 번만”이라는 원칙이 중요한 이유가 있습니다. 테스트 성능을 보고 모델을 수정한 뒤 다시 테스트 성능을 측정하는 과정을 반복하면, 사실상 테스트 데이터로 튜닝을 한 셈이 됩니다. 그러면 테스트 성능은 더 이상 “새로운 데이터에서의 성능”을 의미하지 않게 됩니다.

2-2. 데이터 누수(Data Leakage)란 무엇인가

데이터 누수는 테스트(또는 미래) 데이터의 정보가 모델 학습 과정에 의도치 않게 스며드는 현상입니다. 모델이 실제보다 좋아 보이게 만들고, 실무 배포 후에야 성능이 급락하는 형태로 나타납니다.

가장 흔한 누수 유형은 다음과 같습니다.

1. 전처리 순서 오류: 표준화(StandardScaler)를 훈련/테스트 분할 전에 전체 데이터로 fit 하면, 테스트 데이터의 평균과 분산 정보가 훈련 과정에 흘러들어갑니다. 반드시 훈련 데이터로만 fit 하고, 테스트 데이터에는 transform 만 적용해야 합니다.
2. 변수 선택 순서 오류: target 과의 상관관계를 기준으로 변수를 선택할 때, 전체 데이터(테스트 포함)를 기준으로 선택하면 누수가 발생합니다. 변수 선택도 훈련 데이터 안에서만 수행해야 합니다.
3. 시간 정보 무시: 시계열 데이터에서 미래 시점 데이터가 과거 예측에 쓰이는 경우. ARIMA 가이드에서 별도로 다룹니다.

올바른 전처리 순서

1. 훈련/테스트 분할 먼저
2. 훈련 데이터로만 StandardScaler.fit()
3. 훈련 데이터에 transform() 적용
4. 테스트 데이터에 transform()만 적용 (fit 다시 하지 않음)

이 순서를 지키지 않으면 테스트 성능이 실제보다 낙관적으로 나타납니다.

R

```
# 올바른 전처리 순서 (R)
library(caret)

# y: 악성=1, 양성 종양=0 (sklearn 원본을 1 - target으로 뒤집은 값)
# 1. 분할 먼저
set.seed(42)
train_idx <- createDataPartition(y, p=0.8, list=FALSE)
X_train <- X[train_idx,]; X_test <- X[-train_idx,]
y_train <- y[train_idx]; y_test <- y[-train_idx]

# 2. 훈련 데이터로만 scaler 학습
preproc <- preProcess(X_train, method=c('center','scale'))
# 3. 각각 transform
X_train_s <- predict(preproc, X_train)
X_test_s <- predict(preproc, X_test) # fit 없이 transform만
```

Python

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True)
y = 1 - y # 원본은 양성 종양=1 → 악성을 1(positive)로 뒤집음

# 1. 분할 먼저
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# 2. 훈련 데이터로만 fit
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)

# 3. 테스트 데이터에는 transform만 (fit 다시 하지 않음)
X_test_s = scaler.transform(X_test)
```

3. 교차검증

3-1. k-fold CV의 원리

교차검증의 문제의식은 단순합니다. 훈련 데이터를 한 번 무작위로 분할하면, 운 좋게 쉬운 검증 세트가 걸릴 수도 있고 운 나쁘게 어려운 검증 세트가 걸릴 수도 있습니다. 이 불확실성을 줄이기 위해 k-fold CV는 훈련 데이터를 k개의 동일한 크기 조각(fold)으로 나눈 뒤, 한 번에 하나씩 검증용으로 쓰고 나머지로 학습하는 과정을 k번 반복합니다. 결과적으로 훈련 데이터의 모든 샘플이 정확히 한 번씩 검증에 쓰이고, k개의 성능 점수를 평균 내어 최종 성능을 추정합니다.

k는 보통 5 또는 10으로 설정합니다. k가 클수록 더 정확한 성능 추정이 가능하지만 계산 비용이 늘어납니다. 데이터가 충분하다면 5-fold, 작다면 10-fold를 권합니다.

3-2. Stratified k-fold — 분류 문제에서 특히 중요

일반 k-fold 는 fold 를 무작위로 나누므로, 어떤 fold 에는 양성 샘플이 많고 어떤 fold 에는 없을 수 있습니다. 분류 문제에서 클래스 비율이 고르지 않은 경우(불균형 데이터), 이 편차가 성능 추정에 큰 영향을 줍니다. Stratified k-fold 는 각 fold 가 원본 데이터의 클래스 비율을 동일하게 유지하도록 분할합니다.

언제 Stratified k-fold 를 써야 하는가

분류 문제에서는 Stratified k-fold 가 기본값이어야 합니다.

특히 클래스 비율이 불균형(양성 10% 이하 등)할 때 일반 k-fold 와 결과 차이가 크게 납니다.

회귀 문제에서는 일반 k-fold 를 사용합니다.

3-3. 시계열 데이터에서의 CV

시계열 데이터에서는 일반 k-fold 를 쓰면 안 됩니다. 미래 데이터로 과거를 예측하는 누수가 발생하기 때문입니다. 대신 항상 과거 → 미래 방향으로만 분할하는 Walk-forward Validation(또는 Time Series Split)을 사용합니다. 자세한 내용은 ARIMA 가이드에서 다룹니다.

3-4. R / Python 코드

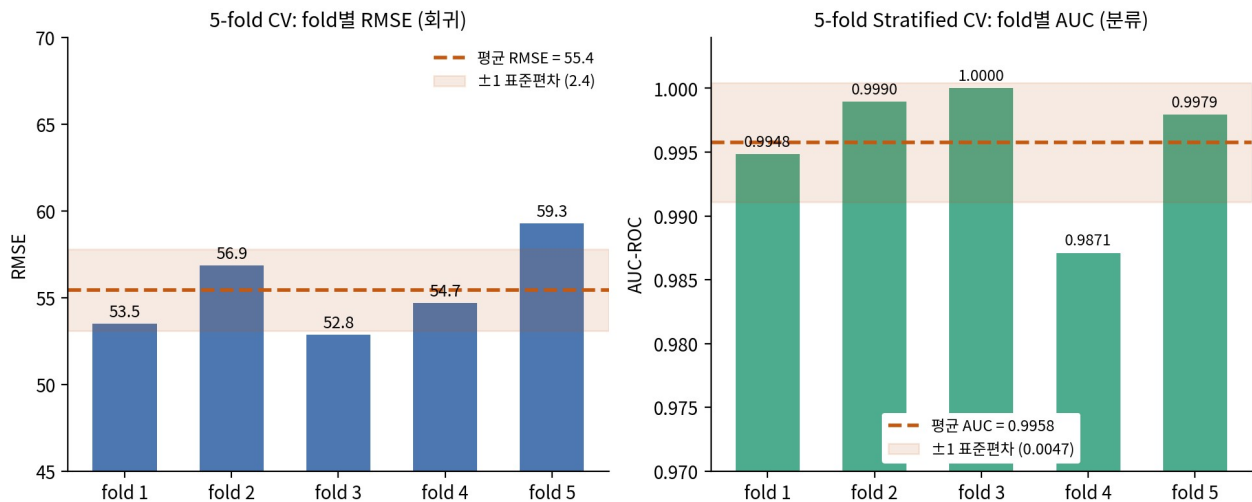


그림 2. 5-fold CV 에서 fold 별 성능 점수. 회귀(왼쪽)는 RMSE 가 52.8~59.3 범위에서 변동하고, 분류(오른쪽)는 AUC 가 0.9871~1.0000 범위로 안정적이다. 표준화는 Pipeline 안에서 fold 마다 훈련 fold 로만 다시 학습했다. 평균 $\pm$ 표준편차로 성능을 요약하는 것이 단일 분할보다 신뢰할 수 있다.

R

```
library(caret)

# 회귀: 5-fold CV
ctrl <- trainControl(method='cv', number=5)
cv_fit <- train(target ~ ., data=train_df, method='ridge',
  preprocess=c('center','scale'),
  trControl=ctrl, metric='RMSE')
print(cv_fit$results)

# 분류: Stratified 5-fold CV
ctrl_cls <- trainControl(method='cv', number=5, classProbs=TRUE,
```

```

summaryFunction=twoClassSummary)
cv_cls <- train(target ~ ., data=train_df, method='glm',
preProcess=c('center','scale'),
trControl=ctrl_cls, metric='ROC')
print(cv_cls$results)

```

Python

```

from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler # 표준화는 Pipeline 안에서 fold마다
다시 학습
import numpy as np

# 회귀: 5-fold CV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
rmse_scores = np.sqrt(-cross_val_score(
    make_pipeline(StandardScaler(), Ridge(alpha=10)), X_train, y_train,
    cv=kf, scoring='neg_mean_squared_error'))
print(f'RMSE: {rmse_scores.mean():.2f} ± {rmse_scores.std():.2f}')

# 분류: Stratified 5-fold CV
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
auc_scores = cross_val_score(
    make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000)),
    X_train, y_train,
    cv=skf, scoring='roc_auc')
print(f'AUC: {auc_scores.mean():.4f} ± {auc_scores.std():.4f}')

```

4. 평가지표 선택하기

4-1. 회귀 지표: RMSE, MAE, R²

회귀 모델의 예측이 실제값에서 얼마나 벗어나는지를 측정하는 지표들입니다.

$$RMSE = \sqrt{(\sum (y_i - \hat{y}_i)^2 / n)} \quad MAE = \sum |y_i - \hat{y}_i| / n$$

- RMSE(평균제곱근오차): 큰 오차에 더 민감합니다(제곱 때문). 이상치가 있을 때 RMSE가 MAE보다 크게 튀는 경향이 있습니다. 원래 단위(예: 원, kg)로 해석됩니다.
- MAE(평균절대오차): 모든 오차를 균등하게 취급합니다. 이상치에 덜 민감하고, 직관적으로 해석하기 쉽습니다.
- R²(결정계수): 모델이 전체 분산 중 얼마를 설명하는지 나타냅니다. 훈련 데이터에서는 0~1 사이 값이지만, 테스트 데이터에서는 평균값으로만 예측하는 것보다 못하면 음수가 될 수도 있습니다. 단위가 없어 해석이 쉽지만, 훈련 데이터 기준 R²는 변수를 추가할수록 자동으로 증가하므로 변수가 많은 모델을 과대평가하기 쉽습니다.

지표	Diabetes Ridge (테스트셋)	해석
RMSE	53.63	예측이 평균적으로 실제값에서 53.6 정도 벗어남
MAE	42.86	예측이 평균적으로 실제값에서 42.9 정도 벗어남
R ²	0.457	모델이 전체 분산의 약 45.7%를 설명함

4-2. 분류 지표: 혼동행렬에서 출발하기

분류 모델의 성능은 예측이 맞고 틀린 경우를 네 가지 유형으로 나누는 혼동행렬(Confusion Matrix)에서 시작합니다.

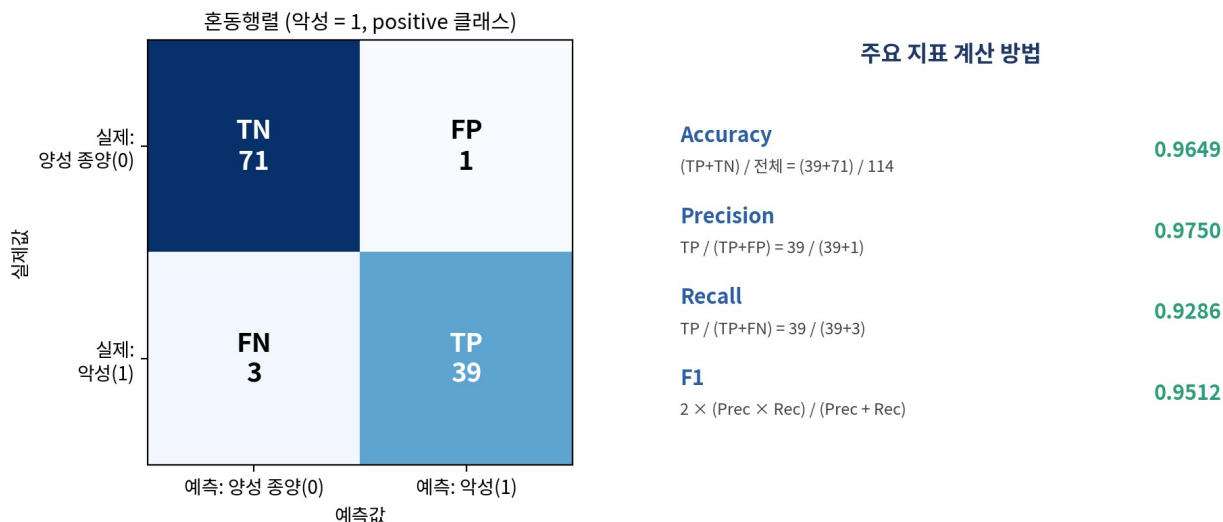


그림 3. Breast Cancer 데이터에 대한 로지스틱 회귀의 혼동행렬(왼쪽)과 주요 지표 계산 방법(오른쪽). TN=71(양성 종양을 양성 종양으로), FP=1(양성 종양을 악성으로 잘못 분류), FN=3(악성을 양성 종양으로 놓침), TP=39(악성을 악성으로 올바르게 분류).

이 예제에서 Precision 은 0.9750(악성으로 예측한 40 건 중 39 건이 실제 악성), Recall 은 0.9286(실제 악성 42 건 중 39 건을 찾아냄)입니다. Accuracy 는 0.9649 로 높아 보이지만, 그 이면에 악성 3 건을 놓친(FN=3) 결과가 숨어 있습니다. 두 지표 중 어느 쪽을 더 중시할지는 문제의 맥락에 따라 달라집니다.

- 암 진단: FN(악성을 양성 종양으로 놓침)이 FP(양성 종양을 악성으로 잘못 분류)보다 훨씬 위험 → Recall 최우선
- 스팸 필터: FP(정상 메일을 스팸으로 잘못 분류)가 더 불편 → Precision 중시
- F1-score 는 Precision 과 Recall 의 조화평균으로, 두 지표의 균형을 하나의 숫자로 요약합니다.

4-3. AUC-ROC 와 Precision-Recall 곡선

Accuracy, Precision, Recall, F1 은 모두 “임계값 0.5 에서 예측 확률을 0 또는 1 로 바꾼 뒤” 계산됩니다. 그런데 임계값을 어떻게 정하느냐에 따라 이 지표들이 달라집니다. 두 모델을 임계값과 무관하게 비교하려면 모든 임계값에 걸친 성능을 곡선으로 요약해야 합니다.

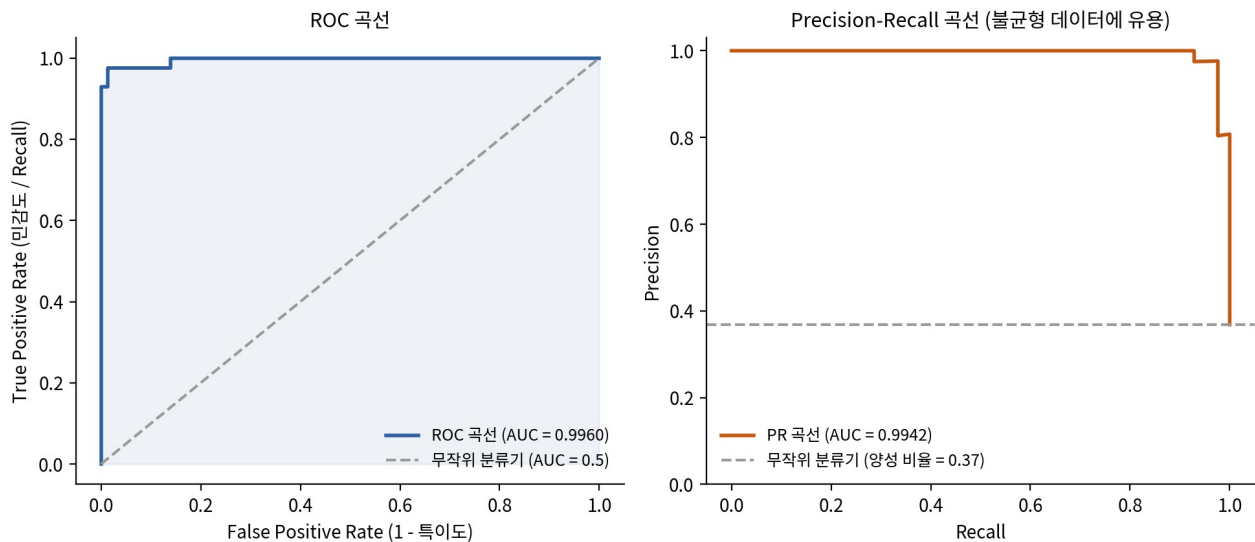


그림 4. ROC 곡선(왼쪽)과 Precision-Recall 곡선(오른쪽). ROC AUC = 0.9960, PR AUC = 0.9942. PR 곡선의 무작위 기준선은 양성 비율(0.37)이다. 두 곡선 모두 모델의 전반적인 분류 능력을 임계값에 독립적으로 요약한다.

- AUC-ROC: 무작위 분류기(AUC=0.5)보다 얼마나 나은지를 나타냅니다. 클래스가 균형잡힌 경우에 적합합니다.
- PR AUC(Precision-Recall 곡선 면적): 불균형 데이터에서 ROC 보다 민감한 지표입니다. 양성 클래스가 매우 드물 때(사기 탐지, 희귀 질환 진단 등) PR AUC 가 ROC 보다 실질적인 성능 차이를 더 잘 드러냅니다.

4-4. 상황별 지표 선택 가이드라인

상황별 평가지표 선택 가이드라인			
상황	추천 지표	피해야 할 지표	이유
불균형 분류 (사기탐지, 진단)	Recall, F1, PR-AUC	Accuracy	양성이 1%여도 Accuracy 99% 가능
비용이 비대칭 (암 진단 등)	Recall 우선	Precision 우선	FN이 FP보다 훨씬 위험
임계값 결정 전 (모델 비교)	AUC-ROC	Fixed Threshold 지표	임계값 독립적 비교 가능
회귀 이상치 민감	MAE	RMSE	RMSE는 큰 오차에 제곱 패널티

그림 5. 상황별 평가지표 선택 가이드라인. 빨간색으로 표시된 “피해야 할 지표”는 그 상황에서 오해를 유발할 수 있는 지표들이다.

가장 많이 범하는 실수는 불균형 분류 문제에서 Accuracy를 주요 지표로 쓰는 것입니다. 양성 비율이 1%인 데이터셋에서 모든 샘플을 음성으로 예측하는 모델의 Accuracy는 99%입니다. 이 모델은 양성을 하나도 잡지 못하는 완전히 쓸모없는 모델이지만, Accuracy만 보면 “매우 좋은 모델”처럼 보입니다. 불균형 데이터 분석 가이드에서 이 문제를 더 자세히 다룹니다.

R

```
# 분류 지표 계산 (R)
library(caret); library(pROC)

# 혼동행렬 및 주요 지표
cm <- confusionMatrix(factor(y_pred), factor(y_test), positive='1')
print(cm) # Accuracy, Precision(Pos Pred Value), Recall(Sensitivity), F1

# AUC-ROC
roc_obj <- roc(y_test, y_prob)
print(auc(roc_obj))
```

Python

```
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_auc_score, average_precision_score)

# 분류 보고서 (Precision, Recall, F1 한번에)
print(classification_report(y_test, y_pred))

# AUC-ROC와 PR AUC
print('AUC-ROC:', roc_auc_score(y_test, y_prob))
print('PR AUC:', average_precision_score(y_test, y_prob))
```

5. 모델 평가 실전 체크리스트

다음 체크리스트는 모델 평가 과정에서 흔히 놓치는 항목들을 정리한 것입니다.

1. 훈련/테스트 분할은 전처리보다 먼저 했는가?
2. 표준화, 변수 선택 등 모든 전처리는 훈련 데이터만으로 학습했는가?
3. 분류 문제에서 Stratified k-fold 를 사용했는가?
4. 성능을 단일 분할 결과가 아닌 CV 평균 $\pm$ 표준편차로 보고하고 있는가?
5. 데이터의 불균형 정도와 비즈니스 맥락에 맞는 지표를 선택했는가?
6. 테스트 데이터를 모델 개발 중에 들여다보지 않았는가?

*이 가이드는 교차검증의 원리, 데이터 누수 방지, 회귀·분류 평가지표의 선택 기준을 다루었습니다.
하이퍼파라미터 탐색(Grid Search, Random Search, Bayesian Optimization)은 이 가이드의 범위를
벗어나며, 앙상블 기법 가이드에서 Random Forest·GBM 의 튜닝 문맥에서 다룹니다.*