교차 검증 (Cross-Validation)

- 모델의 일반화 성능을 더 안정적이고 신뢰성 있게 평가하기 위한 방법
- 훈련 세트와 테스트 세트를 한 번만 분할하는 방식은 데이터가 어떻게 분할되었는지에 따라, 평가 결과가 우연히 좋거나 나쁘게 나올 수 있는 한계가 존재
- 교차 검증은 이 문제를 해결하기 위해 훈련 데이터를 여러 개의 다른 부분 집합으로 나누어, 여러 번의 훈련과 평가를 반복하는 방식입니다.

가장 널리 사용되는 교차 검증 방법은 **K-Fold 교차 검증 (K-Fold Cross-Validation)** 입니다.

1. 전체 훈련 데이터를 비슷한 크기의 K개의 부분 집합(fold)으로 나눕니다.
2. 첫 번째 fold를 검증 세트(validation set)로 사용하고, 나머지 K-1개의 fold를 훈련 세트로 사용하여 모델을 학습하고 평가합니다.
3. 두 번째 fold를 검증 세트로, 나머지 K-1개를 훈련 세트로 사용하여 두 번째 학습과 평가를 수행합니다.
4. 이 과정을 K번 반복하여, 모든 fold가 한 번씩 검증 세트로 사용되도록 합니다.
5. 최종적으로 K개의 평가 점수(e.g., 정확도, MSE)의 평균을 내어 모델의 최종 성능으로 삼습니다.

적용 가능한 상황

- 모델의 성능을 객관적이고 안정적으로 평가하고 싶을 때 사용합니다.
- 데이터의 양이 많지 않아 훈련/검증/테스트 세트로 나누기 부담스러울 때, 데이터를 더 효율적으로 사용하기 위해 적용합니다.
- `GridSearchCV`나 `RandomizedSearchCV`와 같은 하이퍼파라미터 튜닝 과정에서, 각 하이퍼파라미터 조합의 성능을 신뢰성 있게 평가하기 위해 내부적으로 사용됩니다.

예제 데이터

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target
dt_clf = DecisionTreeClassifier(random_state=42)
```

1. K-Fold

- **용도**: 일반적인 회귀 또는 분류 문제에서 사용되는 가장 기본적인 교차 검증 방법
- **주의사항**
  - 데이터를 순차적으로 K개로 분할함
  - 만약 데이터가 특정 순서로 정렬되어 있다면, 반드시 `shuffle=True` 옵션을 사용하여 데이터를 무작위로 섞어야 함
  - 그렇지 않으면 각 fold의 데이터 분포가 편향될 수 있음

- 분류 문제, 특히 클래스가 불균형한 경우에는 `K-Fold`보다 `Stratified K-Fold` 사용을 강력히 권장

```
# KFold 클래스 하이퍼파라미터
# n_splits: fold의 개수 (K). (기본값=5)
# shuffle: fold를 나누기 전 데이터를 섞을지 여부. (기본값=False)
# random_state: shuffle=True일 때 재현성을 위한 시드 값.
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
cv_accuracies_kfold = []
fold_idx = 1

# KFold 객체의 split() 메서드는 훈련/검증용 인덱스를 반환
for train_index, val_index in kfold.split(X):
    # 인덱스를 이용해 훈련/검증 데이터 분할
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # 모델 학습 및 예측
    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_val)

    # 정확도 계산 및 저장
    accuracy = accuracy_score(y_val, pred)
    cv_accuracies_kfold.append(accuracy)
    print(f"K-Fold {fold_idx}차 검증 정확도: {accuracy:.3f}")
    fold_idx += 1
'''
K-Fold 1차 검증 정확도: 1.000
K-Fold 2차 검증 정확도: 0.967
K-Fold 3차 검증 정확도: 0.933
K-Fold 4차 검증 정확도: 0.933
K-Fold 5차 검증 정확도: 0.933
'''

print(f"\nK-Fold 평균 검증 정확도: {np.mean(cv_accuracies_kfold):.3f}") # 0.953
```

2. Stratified K-Fold

- **용도**: 분류 문제, 특히 클래스 분포가 불균형한 데이터셋에 사용
- **주의사항**
  - 각 fold를 생성할 때, 원본 데이터의 클래스 비율을 그대로 유지하도록 데이터를 분할(층화 샘플링)
  - 각 반복(iteration)에서 훈련 세트와 검증 세트의 클래스 분포가 원본 데이터와 유사하게 유지되어, 더 안정적이고 편향 없는 성능 평가가 가능

```
# StratifiedKFold는 split() 메서드에 X와 함께 y(타겟)를 반드시 전달해야 함
skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_accuracies_skfold = []
fold_idx = 1
```

```
for train_index, val_index in skfold.split(X, y):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_val)

    accuracy = accuracy_score(y_val, pred)
    cv_accuracies_skfold.append(accuracy)
    print(f"Stratified K-Fold {fold_idx}차 검증 정확도: {accuracy:.3f}")
    fold_idx += 1
'''
Stratified K-Fold 1차 검증 정확도: 1.000
Stratified K-Fold 2차 검증 정확도: 0.967
Stratified K-Fold 3차 검증 정확도: 0.933
Stratified K-Fold 4차 검증 정확도: 0.967
Stratified K-Fold 5차 검증 정확도: 0.900
'''

print(f"\nStratified K-Fold 평균 검증 정확도: {np.mean(cv_accuracies_skfold):.3f}")
# 0.953
```

3. cross_val_score()

```
# cross_val_score() 함수를 이용한 간편한 교차 검증
# cross_val_score(모델, 특성 데이터, 타겟 데이터, 평가 지표, 교차 검증 폴드 수)
# cv 인자에 KFold, StratifiedKFold 객체를 전달할 수도 있음
scores = cross_val_score(dt_clf, X, y, scoring='accuracy', cv=5)
print(f"\ncross_val_score()를 이용한 교차 검증 정확도: {scores}")
# cross_val_score()를 이용한 교차 검증 정확도:
# [0.96666667 0.96666667 0.9        0.93333333 1.        ]
print(f"cross_val_score() 평균 정확도: {np.mean(scores):.3f}") # 0.953

# 분류 문제에서는 cv에 정수만 입력해도 기본적으로 StratifiedKFold를 사용함
scores_strat = cross_val_score(dt_clf, X, y, scoring='accuracy', cv=skfold)
print(f"\ncross_val_score() (Stratified) 평균 정확도: {np.mean(scores_strat):.3f}")
# 0.953
```

결과 해석 방법

- **개별 검증 결과**
  - 각 fold에서 나온 평가 점수들을 통해 모델 성능의 변동성을 파악할 수 있습니다.
  - 만약 점수들의 편차가 크다면 모델이 데이터 분할 방식에 민감하게 반응하는 불안정한 모델일 수 있습니다.
- **평균 검증 결과**
  - 모든 fold의 평가 점수를 평균한 값으로, 단일 분할보다 훨씬 안정적이고 일반화된 모델의 성능 추정치로 사용됩니다.
  - 이 평균 점수를 기준으로 여러 모델의 성능을 비교하거나 하이퍼파라미터를 선택합니다.

장단점 및 대안

- **장점**:
  - 데이터를 한 번만 분할하는 것보다 모델의 성능을 훨씬 안정적이고 신뢰성 있게 평가할 수 있습니다.
  - 모든 데이터가 훈련과 검증에 한 번씩은 사용되므로, 데이터를 효율적으로 활용할 수 있습니다.
- **단점**:
  - 모델을 K번 훈련시켜야 하므로, 훈련 시간이 오래 걸리는 모델의 경우 전체 평가 시간이 길어집니다.
- **대안**:
  - **Leave-One-Out Cross-Validation (LOOCV)**: K를 전체 데이터 샘플 수(N)와 동일하게 설정하는 극단적인 경우입니다. 데이터 하나만 검증 세트로 사용하고 나머지를 모두 훈련에 사용합니다. 데이터가 매우 적을 때 유용하지만, 계산 비용이 매우 큽니다.
  - **시간 분할 교차 검증 (Time Series Split)**: 시계열 데이터의 경우, 미래의 데이터로 과거의 데이터를 예측하는 상황을 막기 위해 특별한 방식의 교차 검증이 필요합니다. `TimeSeriesSplit`은 훈련 세트가 항상 검증 세트보다 시간적으로 앞서도록 데이터를 분할합니다.