검증과 하이퍼 파라미터 튜닝

- 검증 데이터 세트
- 교차 검증

교차검증cross Validation

교차 검증의 목적

- **교차검증**Cross Validation
  - 안정적인 검증 점수를 얻을 수 있음
  - 더 많은 데이터 세트를 훈련에 사용할 수 있음

교차 검증 함수 사용

cross_val_score(): test_score값만 반환

교차 검증의 최종 점수: test_score키의 5개 값

교차 검증을 수행하면 입력한 모델에서 얻을 수 있는 최상의 검증 점수를 가늠.

Kfold 분할기 : 교차검증을 할 때 훈련세트를 섞으려면 splitter(분할기)를 지정

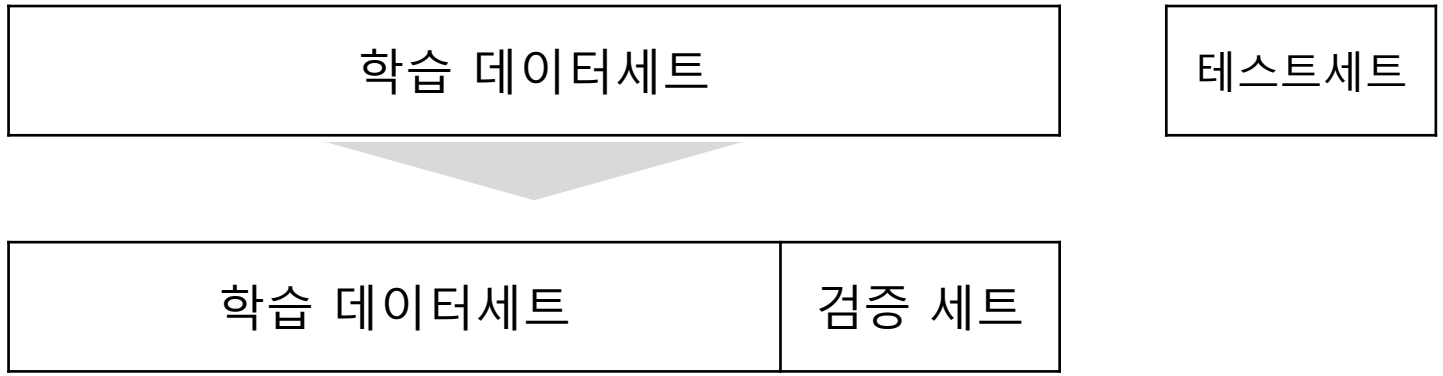
기본 분할기:

회귀모델: KFold

분류모델: StratifiedKFold

교차 검증cross-validation

- 기본절차
  1. 데이터를 나눔(학습용,테스트용)
  2. 모델의 테스트성능 기록
  3. 교차 검증의 매 단계마다 다른 파티션으로 위의 작업수행
  4. 최종성능: 매 단계의 테스트 성능 평균 계산
- 기법 : K폴드 교차검증K-fold CV, leave-one-out 교차검증 , Leave-p-out 교차검증Leave-p-out CV, Shuffle split 교차검증



K-폴드 교차검증K-fold CV

데이터를 무작위로 중복없이 K개의 동일한 크기의 폴드로 나눔(예:3, 5, 10)

K-1겹으로 모델을 훈련하고 나머지 하나로 성능을 평가함(각 폴드를 테스트세트로 한 번씩 사용)

즉, K번 반복하므로 K개의 서로 다른 모델을 얻을 수 있음(K폴드(겹))

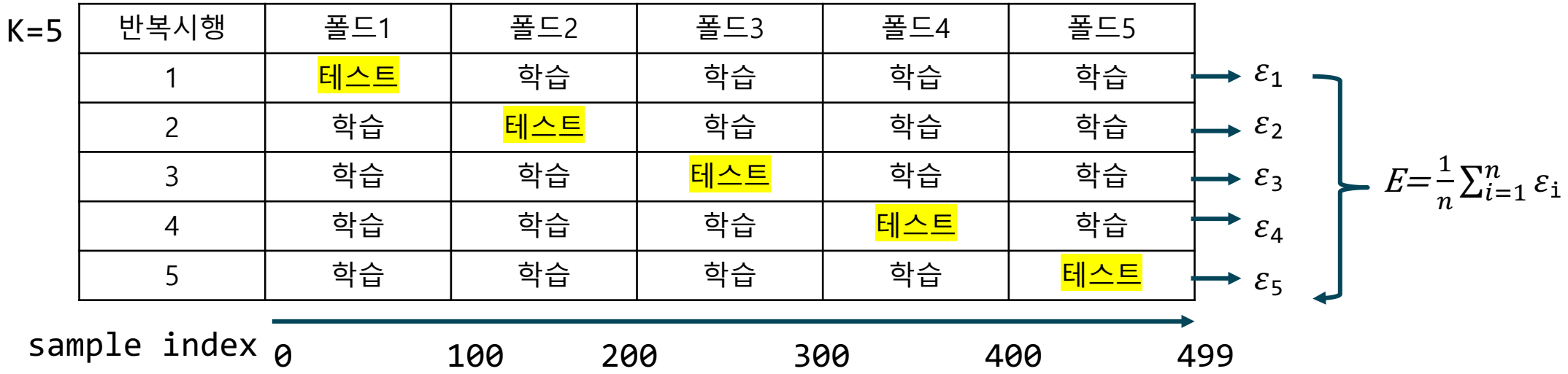
홀드아웃교차 검증의 단점(데이터를 분할 방법에 따라 평가 결과 상이) 교정

각각의 폴드에서 얻은 성능을 기반으로 최종적으로 모델 성능의 평균을 계산

홀드아웃 방법보다 데이터 분할에 덜 예민한 성능 평가 가능

K폴드 교차 검증은 중복을 허락하지 않기 때문에 모든 샘플이 검증에 1회씩 사용됨

K가 클수록 시간이 오래 걸림



K폴드 교차검증 실습(1)

- sklean.model_selection 모듈의 KFold() 사용(with for loop)

1. 폴드를 분리할 객체 생성

```
sklearn.model_selection.KFold(  n_splits = 분리할 폴드의 개수, 기본값은 3
                                shuffle = 데이터를 섞어서 분리할 것인지 여부, 기본값은 False,
                                random_state = 동일한 집합으로 생성할 수 있는 규칙을 정하는 정수값)
```

```
1 from sklearn.model_selection import KFold
2 kfold = KFold(5) # 객체 생성
```

K폴드 교차검증 실습(1)

2. 데이터를 준비하고 회귀 모델 객체를 생성

```
1  from sklearn.datasets import load_diabetes
2  from sklearn.linear_model import LinearRegression
3
4  diab = load_diabetes()
5  X = diab.data
6  y = diab.target
7
8  lr = LinearRegression()
```

K폴드 교차검증 실습(1)

3. split() 함수를 호출하여 폴드별로 분리될 행 인덱스 세트를 구함

```
sklearn.model_selcltion.KFold.split(X = 분리할 특성 데이터셋,
                                    y = 분리할 클래스 데이터셋, 기본값은 None
                                    groups = 분리할 샘플에 붙이는 그룹 레이블 기본값은 None)
```

```python
from sklearn.metrics import r2_score

r2_scores = []

for train_idx, test_idx in kfold.split(X):
    # 인덱스 번호를 받아 레코드 분리
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    reg = lr.fit(X_train, y_train)

    y_pred = reg.predict(X_test)
    r2_scores.append(r2_score(y_test, y_pred))
```

K폴드 교차검증 실습(1)

4. 각 폴드별 결정 계수와 전체 평균 결정 계수값을 확인.

```
import numpy as np

for i, r2 in enumerate(r2_scores):
    print(i, ": R2 - {:.3f}".format(r2))
print("average R2: ", np.round(np.mean(r2_scores), 3))
```

```
0 : R2 - 0.430
1 : R2 - 0.523
2 : R2 - 0.483
3 : R2 - 0.427
4 : R2 - 0.550
average R2:  0.482
```

K폴드 교차검증 실습(2)

- sklean.model_selection 모듈의 cross_val_score() 사용

```
sklearn.model_selection.cross_val_score(평가할 모델 객체 추정기,
                                        데이터 특성(독립변수) 세트, 데이터 레이블(종속변수) 세트,
                                        scoring = 성능 평가 함수 이름(명시하지 않으면 추정기에 따름),
                                        cv = 교차 검증을 수행할 폴드의 개수 또는 KFold 객체, 기본값은 5)
```

```python
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

diab = load_diabetes()
X = diab.data
y = diab.target

lr = LinearRegression()

r2_scores = cross_val_score(lr, X, y, cv=5)

print("R2: ", np.round(r2_scores, 3))
print("average R2: ", np.round(np.mean(r2_scores), 3))
```

검증 세트를 직접 떼어내는 KFold()와 달리 훈련 세트 전체를 함수로 전달함

K폴드 교차검증 실습(3)

- cross_val_score() with shuffling : 폴드 분리 방법에 따른 성능 지표 변화

```
1  from sklearn.datasets import load_diabetes
2  from sklearn.linear_model import LinearRegression
3  from sklearn.model_selection import KFold
4  from sklearn.model_selection import cross_val_score
5
6  diab = load_diabetes()
7  X = diab.data
8  y = diab.target
9
10 lr = LinearRegression()
11
12 kfold = KFold(3, shuffle=True, random_state=0)
13 r2_scores =cross_val_score(lr, X, y, cv=kfold)
14
15 print("R2: ", np.round(r2_scores, 3))
16 print("average R2: ", np.round(np.mean(r2_scores), 3))
```

```
R2:  [0.404 0.521 0.544]
average R2:  0.49
```

cross_val_score()에서 폴드 방법을 지정할 수 없음.
폴드 객체를 생성할 때 폴드 분리 방법 지정

폴드 분리를 어떻게 하느냐에 따라 검증 결과가 달라지고, 이에 따라 성능 지표 역시 변화하는 것을 확인할 수 있다.

- cross_validate(평가할 모델, 훈련세트 전체(X), 훈련세트전체(y))
- 반환값 – fit_time, score_time, test_score키를 가진 딕셔너리 반환

하이퍼파라미터 튜닝

- GridSearchCV
- RandomSearchCV

하이퍼 파라미터 튜닝

모델 파라미터:머신러닝 모델이 학습하는 파라미터

하이퍼파라미터: 모델이 학습할수 없어서 사용자가 지정해야만 하는 파라미터

하이퍼파라미터 튜닝

1. 기본값을 그대로 모델에 사용

2. 검증세트의 점수나 교차 검증을 통해서 매개변수를 조금씩 바꾸어봄.(예, 1 ~ 6개)

3 AutoML : 사람의 개입 없이 하이퍼라파미터 튜닝을 자동으로 수행하는 기술

고려 사항:

1. 결정트리 모델에서 max_depth값을 찾은 경우, 그다음 max_max_depth값을 최적의 값으로 고정하고 min_samples_split을 바꾸어가며 최적을 값을 찾는 것이 아니라, 두 매개변수를 동시에 바꿔가며 최적을 찾는 것

2. 매개변수가 많아지면 문제는 더 복잡해진다.

3. 파이썬의 for반복문으로 찾을 수도 있지만, GridSerach()를 사용하여 효과적으로 찾을 수 있다

[GriSearchCV class]

- 하이퍼파라미터 탐색과 교차 검증을 한 번에 수행
- 별도로 cross_validate()을 호출할 필요가 없음

[실습] 하이퍼 파라미터 튜닝과 GridSearchCV

- 기본 매개 변수를 사용한 결정 트리 모델에서 min_impurity_decrease 매개변수의 최적값 찾기

```
from sklearn.model_selection import GridSearchCV
params={'min_impurity_decrease':[0.0001, 0.0002,0.0003,0.0004,0.0005]}

gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)
# 결정트리 클래스의 객체를 생성하자마자 바로 전달
# cv 기본값 5
# min_impurity_decrese값마다 5번의 교차검증 25개의 모델을 훈련
# n_jobs= 병렬 실행에 사용할 CPU 코어 수(기본값 1, 모든 코어 사용: -1)

gs.fit(X_train, y_train)
```

```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'min_impurity_decrease': [0.0001, 0.0002, 0.0003,
                                                   0.0004, 0.0005]})
```

랜덤 서치

랜덤 서치: 매개변수를 샘플링할 수 있는 확률분포객체를 전달하여 최적의 매개변수를 찾는 방법

매개변수의 값이 수치일 때 값의 범위나 간격을 미리 정하기 어려울 때,

또는 너무 많은 매개 변수 조건이 있어서 그리드서치 수행시간이 오래 걸리는 경우