교차 검증과 그리드 서치

구글 코랩에서 실행하기

검증 세트

In [1]:
```
import pandas as pd

wine = pd.read_csv('https://bit.ly/wine_csv_data')
```

In [2]:
```
data = wine[['alcohol', 'sugar', 'pH']].to_numpy() # x(features)
target = wine['class'].to_numpy() # y
```

In [3]:
```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(
    data, target, test_size=0.2, random_state=42)
```

In [4]:
```
sub_input, val_input, sub_target, val_target = train_test_split(
    train_input, train_target, test_size=0.2, random_state=42)
```

In [5]:
```
print(sub_input.shape, val_input.shape) # 훈련 세트와 검증 세트
```

```
(4157, 3) (1040, 3)
```

In [6]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt.fit(sub_input, sub_target)

print(dt.score(sub_input, sub_target))
print(dt.score(val_input, val_target)) # overfitting
```

```
0.9971133028626413
0.864423076923077
```

교차 검증

In [8]:
```
from sklearn.model_selection import cross_validate

scores = cross_validate(dt, train_input, train_target) # cross_validate에 인수로 평가 모델 객체, train_input, train_target를 전달
print(scores)
# 교차 검증의 최종 점수는 test_score 키에 담긴 5개의 점수를 평균하여 얻을 수 있음
```

```
{'fit_time': array([0.01599765, 0.00840855, 0.0075953 , 0.00791526, 0.00715709]), 'score_time': array([0.00259399, 0.00116324, 0.00112748, 0.0011251 , 0.00107455]), 'test_scor
e': array([0.86923077, 0.84615385, 0.87680462, 0.84889317, 0.83541867])}
```

In [9]:
```
import numpy as np

print(np.mean(scores['test_score'])) # 교차 검증의 최종 점수 출력
```

```
0.855300214703487
```

In [10]:
```
# cross_validate()는 훈련 세트를 섞어 폴드를 나누지 않음, 앞서 train_test_split()함수로 데이터를 섞은 후 훈련 세트를 준비했기 때문에
# 따로 섞을 필요가 없음
# 하지만, 교차 검증을 시행한다면 분할기를 지정해야 함
from sklearn.model_selection import StratifiedKFold # 분류 모델일 경우 stratifiedkfold를 사용

scores = cross_validate(dt, train_input, train_target, cv=StratifiedKFold())
print(np.mean(scores['test_score'])) # 최종 점수 출력
```

```
0.855300214703487
```

In [11]:
```
splitter = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
# 훈련 세트를 섞은 후 10-폴드 교차 검증 수행
scores = cross_validate(dt, train_input, train_target, cv=splitter)
print(np.mean(scores['test_score'])) # 최종 점수 출력
```

```
0.8574181117533719
```

하이퍼파라미터 튜닝

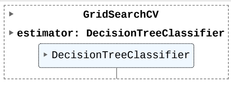
In [12]:
```
from sklearn.model_selection import GridSearchCV
# 사이킷런의 그리드 서치를 이용하여 매개변수의 최적 값을 조정
# GridSearchCV 클래스는 하이퍼파라미터 탐색과 교차 검증을 한 번에 수행

params = {'min_impurity_decrease': [0.0001, 0.0002, 0.0003, 0.0004, 0.0005]}
```

In [13]:
```
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)
# GridSearchCV의 객체 gs를 생성하며, 탐색 대상 모델과 params 변수를 인수로 전달
# n_jobs=-1로 시스템에 있는 모든 CPU코어를 사용할 수 있도록 지정
```

In [15]:
```
gs.fit(train_input, train_target)
# gs 객체에 fit() 메서드를 호출하여 min_impurity_decrease값을 바꿔가며 5번 실행
# GridsearchCV의 cv 매개변수 기본 값을 5로 min_impurity_decrease 값마다 5폴드 교차 검증을 수행
# 총 25번 수행
```

Out[15]:



In [16]:
```
dt = gs.best_estimator_ # 25개의 모델 중에서 검증 점수가 가장 높은 모델이 best_Estimator_ 속성에 저장
print(dt.score(train_input, train_target))
```

```
0.9615162593804117
```

In [ ]:
```
print(gs.best_params_) # 그리드 서치로 찾은 최적의 매개변수는 best_params_ 속성에 저장
```

```
{'min_impurity_decrease': 0.0001}
```

In [17]:
```
print(gs.cv_results_['mean_test_score'])
# 각 매개변수에서 수생한 교차 검증의 평균 점수는 cv_results_ 속서의 "mean_test_Score" zldp wjwkd
```

```
[0.86819297 0.86453617 0.86492226 0.86780891 0.86761605]
```

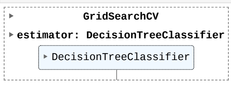
In [18]:
```
best_index = np.argmax(gs.cv_results_['mean_test_score']) # argmax 함수를 통해 가장 큰 값의 인덱스를 추출하여 best_index에 저장
print(gs.cv_results_['params'][best_index])
```

```
{'min_impurity_decrease': 0.0001}
```

In [19]:
```
params = {'min_impurity_decrease': np.arange(0.0001, 0.001, 0.0001), # 노드를 분할하기 위한 불순도 감소 최소량을 지정
          'max_depth': range(5, 20, 1), # 트리의 깊이
          'min_samples_split': range(2, 100, 10) # 노드를 나누기 위한 최소 샘플 수
          }
# 9*5*10*5 = 6750개의 모델의 조합을 시행
```

In [20]:
```
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)
gs.fit(train_input, train_target)
```

Out[20]:



In [21]:
```
print(gs.best_params_)
```

```
{'max_depth': 14, 'min_impurity_decrease': 0.0004, 'min_samples_split': 12}
```

In [22]:
```
print(np.max(gs.cv_results_['mean_test_score']))
```

```
0.8683865773302731
```

랜덤 서치

- 매개변수의 값이 수치일 때 값의 범위나 간격을 미리 정하기 어려울 수 있음
- 혹은, 너무 많은 매개변수 조건이 있어 그리드 서치 수행 시간이 오래 걸릴 수 있음

In [23]:
```
from scipy.stats import uniform, randint
```

In [24]:
```
rgen = randint(0, 10) # 0~10 사이의 범윌르 갖는 randint 객체 생성
rgen.rvs(10) # 10개의 숫자를 샘플링
```

Out[24]:
```
array([9, 6, 9, 2, 0, 9, 2, 7, 0, 8])
```

In [25]:
```
np.unique(rgen.rvs(1000), return_counts=True) # 1000개의 숫자를 샘플링해서 각 숫자의 개수를 출력
```

Out[25]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([ 86, 108,  95,  92,  93, 106, 102,  92,  97, 129]))
```

In [26]:
```
ugen = uniform(0, 1) # 0~1사이의 범위를 갖는 uniform 객체 생성
ugen.rvs(10) # 10개의 숫자를 샘플링
```

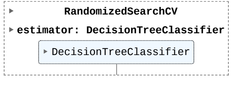
Out[26]:
```
array([0.61272857, 0.46821348, 0.67567833, 0.1128363 , 0.50939157,
       0.44117648, 0.97468117, 0.99318544, 0.632524  , 0.25803874])
```

In [27]:
```
params = {'min_impurity_decrease': uniform(0.0001, 0.001),
          'max_depth': randint(20, 50),
          'min_samples_split': randint(2, 25),
          'min_samples_leaf': randint(1, 25),
          }
```

In [32]:
```
from sklearn.model_selection import RandomizedSearchCV

gs = RandomizedSearchCV(DecisionTreeClassifier(random_state=42), params,
                        n_iter=100, n_jobs=-1, random_state=42)
# 매개변수 n_iter를 통해 100번을 샘플링하여 교차 검증을 수행하여 최적의 매개변수 조합을 찾음
gs.fit(train_input, train_target)
```

Out[32]:



In [33]:
```
print(gs.best_params_)
```

```
{'max_depth': 39, 'min_impurity_decrease': 0.00034102546602601173, 'min_samples_leaf': 7, 'min_samples_split': 13}
```

In [30]:
```
print(np.max(gs.cv_results_['mean_test_score']))
```

```
0.8695428296438884
```

In [31]:
```
dt = gs.best_estimator_

print(dt.score(test_input, test_target))
# 테스트 세트 점수는 검증 세트에 대한 점수보다 조금 작은 것이 일반적
```

```
0.86
```

In [ ]:
```

```