Kfold를 통한 DecisionTreeClf 모델 검증

In [2]:

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

iris = load_iris()
features = iris.data
label = iris.target
dt_clf = DecisionTreeClassifier(random_state=156)

# 5개의 폴드 세트로  분라하는 Kfold 객체와 폴드 세트 별 정확도를 담을 리스트 객체 생성
kfold = KFold(n_splits=5)
cv_accuracy = []

n_iter = 0

# KFold 객체의 split()을 호출하면 폴드별 학습용, 검증용 테스트의 로우 인덱스를 array로 반환
for train_index, test_index in kfold.split(features):
    #kfold.split()으로 반환된 인덱스를 이용해 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]

    # 학습 및 예측
    dt_clf.fit(X_train, y_train)
    y_pred = dt_clf.predict(X_test)
    n_iter += 1

    # 반복 시마다 정확도 측정
    accuracy = np.round(accuracy_score(y_test, y_pred),4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]

    print(f'\n#{n_iter} 교차 검증 정확도 : {accuracy}, 학습 데이터 크기 : {train_size}, 검증 데이터 크기 : {test_si
    print(f'#{n_iter} 검증 세트 인덱스 : {test_index}')
    cv_accuracy.append(accuracy)

# 개별 iteration 별 정확도를 합하여 평균 정확도 계산
print('\n## 평균 검증 정확도 : ', np.mean(cv_accuracy))
```

```
#1 교차 검증 정확도 : 1.0, 학습 데이터 크기 : 120, 검증 데이터 크기 : 30
#1 검증 세트 인덱스 : [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]

#2 교차 검증 정확도 : 0.9667, 학습 데이터 크기 : 120, 검증 데이터 크기 : 30
#2 검증 세트 인덱스 : [30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 54 55 56 57 58 59]

#3 교차 검증 정확도 : 0.8667, 학습 데이터 크기 : 120, 검증 데이터 크기 : 30
#3 검증 세트 인덱스 : [60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
 84 85 86 87 88 89]

#4 교차 검증 정확도 : 0.9333, 학습 데이터 크기 : 120, 검증 데이터 크기 : 30
#4 검증 세트 인덱스 : [ 90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119]

#5 교차 검증 정확도 : 0.7333, 학습 데이터 크기 : 120, 검증 데이터 크기 : 30
#5 검증 세트 인덱스 : [120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
 138 139 140 141 142 143 144 145 146 147 148 149]

## 평균 검증 정확도 :  0.9
```

Stratified Kfold

불균형한 데이터 집합을 위한 K폴드 방식
예를 들면 대출사기 판별하는 경우, 매우 작은 확률이기에 비율을 제대로 반영하기 힘들지만, 매우 중요한 특성이므로 무시할 수 없음
원본 데이터의 레이블 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습과 검증데이터 세트를 분배하여 해결

In [3]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedKFold


iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_df['label'] = iris.target

k_fold = KFold(n_splits=3)
n_iter = 0

for train_index, test_index in kfold.split(iris_df):
    n_iter += 1
    #kfold.split()으로 반환된 인덱스를 이용해 학습용, 검증용 테스트 데이터 추출
    label_train = iris_df['label'].iloc[train_index]
    label_test = iris_df['label'].iloc[test_index]
    print(f'\n## kfold 교차 검증 : {n_iter}')
    print(f'학습 레이블 데이터 분포 : {label_train.value_counts()}')
    print(f'검증 레이블 데이터 분포 : {label_test.value_counts()}')
```

```
## kfold 교차 검증 : 1
학습 레이블 데이터 분포 : 1    50
2    50
0    20
Name: label, dtype: int64
검증 레이블 데이터 분포 : 0    30
Name: label, dtype: int64

## kfold 교차 검증 : 2
학습 레이블 데이터 분포 : 2    50
1    40
0    30
Name: label, dtype: int64
검증 레이블 데이터 분포 : 0    20
1    10
Name: label, dtype: int64

## kfold 교차 검증 : 3
학습 레이블 데이터 분포 : 0    50
2    50
1    20
Name: label, dtype: int64
검증 레이블 데이터 분포 : 1    30
Name: label, dtype: int64

## kfold 교차 검증 : 4
학습 레이블 데이터 분포 : 0    50
1    40
2    30
Name: label, dtype: int64
검증 레이블 데이터 분포 : 2    20
1    10
Name: label, dtype: int64

## kfold 교차 검증 : 5
학습 레이블 데이터 분포 : 0    50
1    50
2    20
Name: label, dtype: int64
검증 레이블 데이터 분포 : 2    30
Name: label, dtype: int64
```

In [4]:

```
skf = StratifiedKFold(n_splits=3)
n_iter = 0

for train_index, test_index in skf.split(iris_df, iris_df['label']):
    # skf는 라벨의 분포에 따라 동일하게 쪼개어주므로, 라벨 변수(y값)도 넣어야 함
    n_iter += 1
    label_train = iris_df['label'].iloc[train_index]
    label_test = iris_df['label'].iloc[test_index]
    print(f'\n## Stratifiedkfold 교차 검증 : {n_iter}')
```

```
    print(f'학습 레이블 데이터 분포 : {label_train.value_counts()}')
    print(f'검증 레이블 데이터 분포 : {label_test.value_counts()}')

## Stratifiedkfold 교차 검증 : 1
학습 레이블 데이터 분포 : 2    34
0    33
1    33
Name: label, dtype: int64
검증 레이블 데이터 분포 : 0    17
1    17
2    16
Name: label, dtype: int64

## Stratifiedkfold 교차 검증 : 2
학습 레이블 데이터 분포 : 1    34
0    33
2    33
Name: label, dtype: int64
검증 레이블 데이터 분포 : 0    17
2    17
1    16
Name: label, dtype: int64

## Stratifiedkfold 교차 검증 : 3
학습 레이블 데이터 분포 : 0    34
1    33
2    33
Name: label, dtype: int64
검증 레이블 데이터 분포 : 1    17
2    17
0    16
Name: label, dtype: int64
```

Stratifiedkfold로 모델학습을 시켜보자~

In [4]:

```
n_iter = 0
cv_accuracy =[]

for train_index, test_index in skf.split(iris_df, iris_df['label']):
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]

    # 학습 및 예측
    dt_clf.fit(X_train, y_train)
    y_pred = dt_clf.predict(X_test)
    n_iter += 1

    # 반복 시마다 정확도 측정
    accuracy = np.round(accuracy_score(y_test, y_pred),4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]

    print(f'\n#{n_iter} 교차 검증 정확도 : {accuracy}, 학습 데이터 크기 : {train_size}, 검증 데이터 크기 : {test_si
    print(f'#{n_iter} 검증 세트 인덱스 : {test_index}')
    cv_accuracy.append(accuracy)

# 개별 iteration 별 정확도를 합하여 평균 정확도 계산
print('\n## 평균 검증 정확도 : ', np.mean(cv_accuracy))
```

```
#1 교차 검증 정확도 : 0.98, 학습 데이터 크기 : 100, 검증 데이터 크기 : 50
#1 검증 세트 인덱스 : [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  50
  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 100 101
 102 103 104 105 106 107 108 109 110 111 112 113 114 115]

#2 교차 검증 정확도 : 0.94, 학습 데이터 크기 : 100, 검증 데이터 크기 : 50
#2 검증 세트 인덱스 : [ 17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  67
  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 116 117 118
 119 120 121 122 123 124 125 126 127 128 129 130 131 132]

#3 교차 검증 정확도 : 0.98, 학습 데이터 크기 : 100, 검증 데이터 크기 : 50
#3 검증 세트 인덱스 : [ 34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  83  84
  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149]

## 평균 검증 정확도 :  0.9666666666666667
```

교차 검증을 보다 간편하게 - cross_val_score

폴드세트를 구성하고 for loop를 통해 학습시키거, 인덱스 추출하는 과정을 단순화 시킨 api
cross_val_score(모델, X, y, scoring = 'accuracy', cv = n) 으로 작성하며,
회귀는 kfold 방식, 분류는 stratifiedkfold 방식으로 구분

In [5]:

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

iris = load_iris()
dt_clf = DecisionTreeClassifier()

data = iris.data
label = iris.target

score = cross_val_score(dt_clf, data, label, scoring='accuracy', cv=3)
print(f'교차 검증 별 정확도 : {np.round(score,4)}')
print(f'평균 검중 정확도 : {np.round(np.mean(score),4)}')
```

```
교차 검증 별 정확도 : [0.98 0.94 0.96]
평균 검중 정확도 : 0.96
```

교차검증과 최적 하이퍼 파라미터 튜닝을 한번에 - GridSearchCV

알고리즘에 사용되는 하이퍼 파라미터를 순차적으로 입력하면서 편리하게 최적의 파라미터를 도출
(Grid는 격자라는 뜻으로 촘촘하게 파라미터를 입력하면서 테스트)

GridSearchCV의 클래스 생성자로 들어가는 주요 파라미터

estimator : classifier, regressor, pipeline 이 사용
pram_grid : key : [리스트 값]을 가지는 딕셔너리가 주어짐, estimator의 튜닝을 위해 파라미터명과 사용될 여러 파라미터값을 지정
scoring : 예측 성능을 측정할 평가 방법을 지정, 보통은 사이킷런의 성능 평가 지표를 지정하는 문자열 (예: 정확도의 경우 'accuracy')로 지정
하나 별도의 성능 평가 지표도 지정할 수 있음
cv : 교차 검증을 위해 분할 되는 학습/테스트 세트의 개수를 지정
refit : 디폴트가 True이며 생성시 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼파라미터로 재학습 시킴

In [6]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

iris = load_iris()

X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)

dt_clf = DecisionTreeClassifier(random_state=156)

grid_parameters = {'max_depth' : [1,2,3,4,5],
                   'min_samples_split' : [2,3,5]
                  }

grid_dtree = GridSearchCV(dt_clf, param_grid=grid_parameters, refit=True)

# grid_parameters를 순차적으로 학습/평가
grid_dtree.fit(X_train, y_train)

# GridSearchCV 결과를 추출하여 DataFrame으로 변환
score_df = pd.DataFrame(grid_dtree.cv_results_)
```

```
score_df[['params','mean_test_score','rank_test_score','split0_test_score','split1_test_score','split2_test_sc
```

Out[6]:

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 1, 'min_samples_split': 2}	0.691667	13	0.708333	0.708333	0.708333
1	{'max_depth': 1, 'min_samples_split': 3}	0.691667	13	0.708333	0.708333	0.708333
2	{'max_depth': 1, 'min_samples_split': 5}	0.691667	13	0.708333	0.708333	0.708333
3	{'max_depth': 2, 'min_samples_split': 2}	0.958333	1	0.958333	1.000000	0.958333
4	{'max_depth': 2, 'min_samples_split': 3}	0.958333	1	0.958333	1.000000	0.958333
5	{'max_depth': 2, 'min_samples_split': 5}	0.958333	1	0.958333	1.000000	0.958333
6	{'max_depth': 3, 'min_samples_split': 2}	0.941667	4	1.000000	1.000000	1.000000
7	{'max_depth': 3, 'min_samples_split': 3}	0.941667	4	1.000000	1.000000	1.000000
8	{'max_depth': 3, 'min_samples_split': 5}	0.941667	4	1.000000	1.000000	1.000000
9	{'max_depth': 4, 'min_samples_split': 2}	0.941667	4	0.958333	1.000000	1.000000
10	{'max_depth': 4, 'min_samples_split': 3}	0.916667	11	0.958333	1.000000	1.000000
11	{'max_depth': 4, 'min_samples_split': 5}	0.941667	4	1.000000	1.000000	1.000000
12	{'max_depth': 5, 'min_samples_split': 2}	0.941667	4	0.958333	1.000000	1.000000
13	{'max_depth': 5, 'min_samples_split': 3}	0.916667	11	0.958333	1.000000	1.000000
14	{'max_depth': 5, 'min_samples_split': 5}	0.941667	4	1.000000	1.000000	1.000000

In [ ]: