lec02. 모델검증

In [1]:
```python
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns


sns.set()

#-------------------- 차트 관련 속성 (한글처리, 그리드) -----------
plt.rcParams['font.family']= 'Malgun Gothic'
plt.rcParams['axes.unicode_minus'] = False

#-------------------- 주피터 , 출력결과 넓이 늘리기 ---------------
# from IPython.core.display import display, HTML
from IPython.display import display, HTML

display(HTML("<style>.container{width:100% !important;}</style>"))
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 100)
pd.set_option('max_colwidth', None)

import warnings
warnings.filterwarnings(action='ignore')
```

In [2]:
```python
from sklearn.model_selection import train_test_split,cross_val_score, GridSearchCV, KFold, StratifiedKFold, cross_validate

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error,  accuracy_score

from sklearn.ensemble     import RandomForestRegressor,RandomForestClassifier
from sklearn.tree         import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn.linear_model import LinearRegression, Ridge, Lasso
# ---- 추가 모델
from sklearn.ensemble     import AdaBoostRegressor, VotingRegressor
from xgboost              import XGBRegressor
from lightgbm             import LGBMRegressor
```

Data Load

In [3]:
```python
from sklearn.datasets import load_iris
```

In [4]:
```python
dataset = load_iris()
dataset.keys()  # dict의 key 확인
```

Out[4]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [5]:
```python
dataset['target']
```

Out[5]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [6]:
```python
dataset['feature_names']
```

Out[6]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [7]:
```python
dataset['data'].shape,dataset['data'][:5]
```

Out[7]:
```
((150, 4),
 array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2]]))
```

In [8]:
```python
df = pd.DataFrame(dataset['data'],
                  columns = ['sl','sw','pl','pw']
                  #columns = dataset['feature_names']
                 )
df['target'] = dataset['target']
df.head()
```

Out[8]:

	sl	sw	pl	pw	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [9]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   sl      150 non-null    float64
 1   sw      150 non-null    float64
 2   pl      150 non-null    float64
 3   pw      150 non-null    float64
 4   target  150 non-null    int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

In [10]:
```python
df['target'].value_counts()
```

Out[10]:
```
0    50
1    50
2    50
Name: target, dtype: int64
```

In [11]:
```python
y=df['target']
X=df.drop('target',axis=1)
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=0.2, random_state=11)
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
score=accuracy_score(y_pred,y_test)
score
```

Out[11]:
```
0.8666666666666667
```

In [12]:
```python
X_train.index, X_test.index
```

Out[12]:
```
(Int64Index([  0, 120, 122,  49,  29, 118, 105,  77,  36,  83,
             ...
              92, 109,  24,  82,  71,  76,  13,  81,  91,  80],
            dtype='int64', length=120),
 Int64Index([112, 145, 133,  56, 111,   9,  65,  15,  30,  63, 119,  62,  84,
             113, 139,  41, 103,  66, 110, 142,  69,   2,  22,  51,  12,  11,
             149,  75,  16,  85],
            dtype='int64'))
```

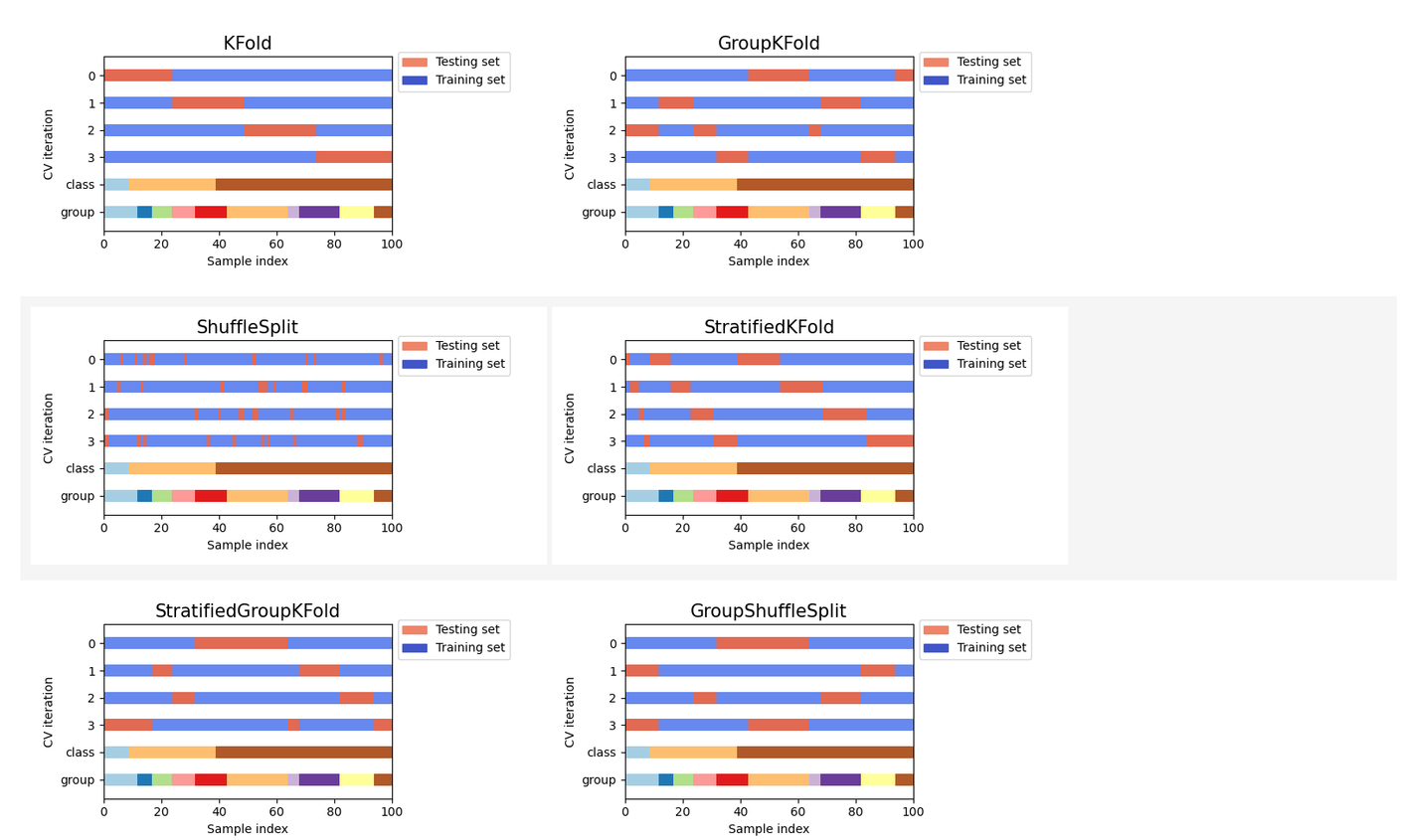
In [13]:
```python
y_train.value_counts(), y_test.value_counts()
```

Out[13]:
```
(0    41
 1    40
 2    39
 Name: target, dtype: int64,
 2    11
 1    10
 0     9
 Name: target, dtype: int64)
```

교차 검증(cross validation)

- GroupKFold : 그룹채로 train, test 나눔 , 그룹의 데이터가 나눠어 train, test로 나누어 질 수 없음
- ShuffleSplit : 무작위로 train, test 나눔 --> test 사이즈 지정 가능, 정해진 cv가 아닌 임의의 사이즈를 주고 싶을 때 사용
- TimeSeriesSplit : 시계열에서 많이 사용

- ref : https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html#sphx-glr-auto-examples-model-selection-plot-cv-indices-py



In [14]:
```python
# 0.9333333333333333 : 기본
# 0.9466666666666667 : KFold
# 0.9333333333333333 : StratifiedKFold
```

KFold()

- 회귀 모델

```
sklearn.model_selection.KFold(n_splits=5,
                              shuffle=False, random_state=None)
```

In [15]:
```python
age=20
gen="남"
print("나이",age,"세","성별",gen)
print(f"나이 : {age}세 성별 : {gen}")
```

```
나이 20 세 성별 남
나이 : 20세 성별 : 남
```

- y=df['target']
- X=df.drop('target',axis=1)

In [29]:
```python
f= KFold(n_splits=5,shuffle=True, random_state=11)

for i, (train_index, test_index) in enumerate(f.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")
    print(df.loc[test_index,'target'].value_counts())
```

```
Fold 0:
  Train: index=[  0   1   3   4   5   6   7   8  10  13  14  17  18  19  20  21  23  24
  25  26  27  28  29  31  32  33  34  35  36  37  38  39  40  42  43  44
  45  46  47  48  49  50  52  53  54  55  57  58  59  60  61  64  67  68
  70  71  72  73  74  76  77  78  79  80  81  82  83  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 104 105 106 107 108 109
 114 115 116 117 118 120 121 122 123 124 125 126 127 128 129 130 131 132
 134 135 136 137 138 140 141 143 144 146 147 148]
  Test:  index=[  2   9  11  12  15  16  22  30  41  51  56  62  63  65  66  69  75  84
  85 103 110 111 112 113 119 133 139 142 145 149]
2    11
1    10
0     9
Name: target, dtype: int64
Fold 1:
  Train: index=[  1   2   3   4   5   7   8   9  10  11  12  13  14  15  16  17  18  19
  22  23  24  25  27  28  30  31  32  33  34  37  38  39  40  41  43  44
  45  47  48  50  51  53  55  56  57  59  62  63  64  65  66  67  68  69
  70  71  72  73  74  75  76  78  79  80  81  82  84  85  87  88  89  90
  91  92  93  94  95  96  97  99 100 101 102 103 107 108 109 110 111 112
 113 114 115 116 117 119 121 123 124 125 126 127 128 129 132 133 134 135
 136 137 138 139 140 142 144 145 146 147 148 149]
  Test:  index=[  0   6  20  21  26  29  35  36  42  46  49  52  54  58  60  61  77  83
  86  98 104 105 106 118 120 122 130 131 141 143]
0    11
2    10
1     9
Name: target, dtype: int64
Fold 2:
  Train: index=[  0   1   2   3   5   6   7   8   9  10  11  12  13  14  15  16  19  20
  21  22  23  24  26  27  29  30  32  34  35  36  37  40  41  42  44  45
  46  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65
  66  67  68  69  70  71  72  74  75  76  77  78  79  80  81  82  83  84
  85  86  87  89  91  92  95  96  98  99 103 104 105 106 107 109 110 111
 112 113 114 115 118 119 120 121 122 124 125 126 128 129 130 131 132 133
 135 137 138 139 141 142 143 144 145 147 148 149]
  Test:  index=[  4  17  18  25  28  31  33  38  39  43  47  48  73  88  90  93  94  97
 100 101 102 108 116 117 123 127 134 136 140 146]
0    12
2    12
1     6
Name: target, dtype: int64
Fold 3:
  Train: index=[  0   2   4   6   9  11  12  13  15  16  17  18  20  21  22  23  24  25
  26  27  28  29  30  31  33  34  35  36  37  38  39  41  42  43  45  46
  47  48  49  51  52  54  56  58  60  61  62  63  64  65  66  67  69  70
  71  73  74  75  76  77  79  80  81  82  83  84  85  86  88  90  91  92
  93  94  96  97  98 100 101 102 103 104 105 106 108 109 110 111 112 113
 114 116 117 118 119 120 122 123 125 127 128 129 130 131 132 133 134 135
 136 138 139 140 141 142 143 145 146 147 148 149]
  Test:  index=[  1   3   5   7   8  10  14  19  32  40  44  50  53  55  57  59  68  72
  78  87  89  95  99 107 115 121 124 126 137 144]
1    12
0    11
2     7
Name: target, dtype: int64
Fold 4:
  Train: index=[  0   1   2   3   4   5   6   7   8   9  10  11  12  14  15  16  17  18
  19  20  21  22  25  26  28  29  30  31  32  33  35  36  38  39  40  41
  42  43  44  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  65  66  68  69  72  73  75  77  78  83  84  85  86  87  88
  89  90  93  94  95  97  98  99 100 101 102 103 104 105 106 107 108 110
 111 112 113 115 116 117 118 119 120 121 122 123 124 126 127 130 131 133
 134 136 137 139 140 141 142 143 144 145 146 149]
  Test:  index=[ 13  23  24  27  34  37  45  64  67  70  71  74  76  79  80  81  82  91
  92  96 109 114 125 128 129 132 135 138 147 148]
1    13
2    10
0     7
Name: target, dtype: int64
```

- y=df['target']
- X=df.drop('target',axis=1)

In [17]:
```python
#------------------------------------------------------------------------------------------
# X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=0.2, random_state=11111)
#------------------------------------------------------------------------------------------
f= KFold(n_splits=5,shuffle=True, random_state=11)
score_list=[]
for i, (train_index, test_index) in enumerate(f.split(X)):

    X_train= X.loc[train_index]
    X_test= X.loc[test_index]
    y_train=y.loc[train_index]
    y_test=y.loc[test_index]

    #     X_train, X_test = X.loc[tridx],  X.loc[teidx]
    #     y_train, y_test = y.loc[tridx],  y.loc[teidx]
#------------------------------------------------------------------------------------------
    model=DecisionTreeClassifier()
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    score=accuracy_score(y_pred,y_test)
    score_list.append(score)
print(np.mean(score_list))
```

```
0.9333333333333333
```

StratifiedKFold()

- 분류 모델
- test에 골고루(이산형에만 해당) 모든 값이 들어가도록 함 --> 반드시 y값 넣어야 함(y를 기준으로 개수를 셈)
- 분류의 train_test_split는 내부적으로 StratifiedKFold 사용

```
sklearn.model_selection.StratifiedKFold(n_splits=5,
                                        shuffle=False, random_state=None)
```

In [18]:
```python
skf= StratifiedKFold(n_splits=5,shuffle=True, random_state=11)

for i, (train_index, test_index) in enumerate(skf.split(X,y)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")
    print(df.loc[test_index,'target'].value_counts())
```

```
Fold 0:
  Train: index=[  0   1   2   3   5   6   7   9  10  11  13  14  16  17  18  19  20  22
  23  24  27  28  29  30  31  32  33  35  36  38  39  40  41  43  44  45
  46  47  48  49  50  51  52  53  54  56  57  58  59  60  61  62  63  64
  65  68  69  70  71  72  74  75  77  78  80  82  83  84  85  86  87  88
  89  90  91  92  93  94  96  97 101 102 103 104 106 107 108 109 110 111
 112 113 114 115 116 117 118 119 120 122 123 124 125 126 127 128 130 131
 132 133 134 137 140 141 143 144 145 146 147 149]
  Test:  index=[  4   8  12  15  21  25  26  34  37  42  55  66  67  73  76  79  81  95
  98  99 100 105 121 129 135 136 138 139 142 148]
0    10
1    10
2    10
Name: target, dtype: int64
Fold 1:
  Train: index=[  0   1   2   3   4   5   6   7   8  11  12  13  14  15  16  17  18  19
  21  22  23  25  26  27  28  29  30  31  32  34  35  36  37  38  39  40
  42  43  47  49  50  51  52  53  54  55  57  58  59  61  62  63  65  66
  67  68  70  71  72  73  74  75  76  78  79  80  81  82  83  84  87  88
  89  90  93  94  95  97  98  99 100 101 102 103 104 105 106 108 109 110
 114 115 117 118 119 120 121 122 125 126 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 148]
  Test:  index=[  9  10  20  24  33  41  44  45  46  48  56  60  64  69  77  85  86  91
  92  96 107 111 112 113 116 123 124 127 147 149]
0    10
1    10
2    10
Name: target, dtype: int64
Fold 2:
  Train: index=[  0   1   2   4   5   7   8   9  10  11  12  13  14  15  16  17  18  20
  21  23  24  25  26  27  29  30  31  33  34  35  36  37  38  41  42  43
  44  45  46  48  50  51  54  55  56  58  59  60  62  63  64  65  66  67
  68  69  71  73  74  75  76  77  79  80  81  82  83  84  85  86  87  88
  91  92  93  94  95  96  98  99 100 101 102 104 105 106 107 108 109 111
 112 113 114 115 116 121 122 123 124 125 126 127 128 129 130 132 135 136
 137 138 139 140 141 142 144 145 146 147 148 149]
  Test:  index=[  3   6  19  22  28  32  39  40  47  49  52  53  57  61  70  72  78  89
  90  97 103 110 117 118 119 120 131 133 134 143]
0    10
1    10
2    10
Name: target, dtype: int64
Fold 3:
  Train: index=[  1   3   4   5   6   8   9  10  11  12  15  16  18  19  20  21  22  23
  24  25  26  28  29  30  31  32  33  34  35  37  39  40  41  42  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  60  61  64  66  67  68
  69  70  72  73  74  76  77  78  79  81  82  83  84  85  86  87  89  90
  91  92  94  95  96  97  98  99 100 101 103 104 105 107 110 111 112 113
 114 116 117 118 119 120 121 122 123 124 127 128 129 130 131 132 133 134
 135 136 138 139 140 141 142 143 146 147 148 149]
  Test:  index=[  0   2   7  13  14  17  27  36  38  43  58  59  62  63  65  71  75  80
  88  93 102 106 108 109 115 125 126 137 144 145]
0    10
1    10
2    10
Name: target, dtype: int64
Fold 4:
  Train: index=[  0   2   3   4   6   7   8   9  10  12  13  14  15  17  19  20  21  22
  24  25  26  27  28  32  33  34  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  52  53  55  56  57  58  59  60  61  62  63  64  65  66
  67  69  70  71  72  73  75  76  77  78  79  80  81  85  86  88  89  90
  91  92  93  95  96  97  98  99 100 102 103 105 106 107 108 109 110 111
 112 113 115 116 117 118 119 120 121 123 124 125 126 127 129 131 133 134
 135 136 137 138 139 142 143 144 145 147 148 149]
  Test:  index=[  1   5  11  16  18  23  29  30  31  35  50  51  54  68  74  82  83  84
  87  94 101 104 114 122 128 130 132 140 141 146]
0    10
1    10
2    10
Name: target, dtype: int64
```

In [19]:
```python
skf= StratifiedKFold(n_splits=5,shuffle=True, random_state=11)
score_list=[]
for i, (train_index, test_index) in enumerate(skf.split(X,y)):

    X_train= X.loc[train_index]
    X_test= X.loc[test_index]
    y_train=y.loc[train_index]
    y_test=y.loc[test_index]

    #     X_train, X_test = X.loc[tridx],  X.loc[teidx]
    #     y_train, y_test = y.loc[tridx],  y.loc[teidx]
#------------------------------------------------------------------------------------------
    model=DecisionTreeClassifier()
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    score=accuracy_score(y_pred,y_test)
    score_list.append(score)
print(np.mean(score_list))
```

```
0.9400000000000001
```

cross_val_score()

- test_score만 리스트로 준다
- 하나의 모델을 가지고 cv번 반복하며 점수를 내줌
- 점수의 list 반환(cv만큼)

```
sklearn.model_selection.cross_val_score(estimator, X, y=None,
scoring=None, cv=None, fit_params=None
groups=None, , pre_dispatch='2*n_jobs', error_score=nan, n_jobs=None, verbose=0)


- estimator : model
- scoring : 점수
- cv : 몇번 반복? = Fold
- fit_params : 제일 좋았던 결과를 저장하여 모델에 적용(True)
- n_jobs : 스레드 개수 지정
    - (None or -1) : 가능한 스레드의 개수 최대로 지정
```

> 기본 개념

- 프로세스 : 실행중인 프로그램
  - 단일프로세스 : 하나의 프로그램만 실행 가능
  - 멀티프로세스 : n개의 프로그램을 동시 실행 가능
- 스레드 : 프로세스 내에서 실제로 작업을 수행하는 주체(작업자)
  - 모든 프로세스에서는 한개 이상의 스레드가 반드시 존재한다
  - 하나의 스레드는 하나의 역할만 수행

In [20]:
```python
model=DecisionTreeClassifier()
score_list=cross_val_score(model,X, y,scoring='accuracy',cv=5)
print(score_list)
print(np.mean(score_list))
```

```
[0.96666667 0.96666667 0.9        1.         1.        ]
0.9666666666666668
```

In [21]:
```python
# StratifiedKFold의 코드와 동일
skf= StratifiedKFold(n_splits=5,shuffle=True, random_state=11)
model=DecisionTreeClassifier(random_state=11)
score_list=cross_val_score(model,X, y,scoring='accuracy',cv=skf)
print(score_list)
print(np.mean(score_list))
```

```
[0.93333333 0.96666667 0.86666667 0.96666667 0.96666667]
0.9400000000000001
```

cross_validate()

- test_score 이외의 것들도 딕셔너리로 준다
  - 점수 : train_score, test_score
  - 시간 : fit_time, score_time

In [22]:
```python
skf= StratifiedKFold(n_splits=5,shuffle=True, random_state=11)
model=DecisionTreeClassifier(random_state=11)
# score_list=cross_val_score(model,X, y,scoring='accuracy',cv=skf)
score_dict=cross_validate(model,X, y,scoring='accuracy',cv=skf) # ,return_train_score=True)
print(score_dict)
print(np.mean(score_dict['test_score']))
```

```
{'fit_time': array([0.00299239, 0.00399017, 0.00310874, 0.00331998, 0.00298333]), 'score_time': array([0.00199509, 0.00199294, 0.0018785 , 0.00300074, 0.00199556]), 'test_score': array([0.93333333, 0.96666667, 0.86666667, 0.96666667, 0.96666667])}
0.9400000000000001
```

GridSearchCV()

- S.KFold() + 모델 Hyper-parameter tunning
- 데이터를 주지 않기 때문에 case by case를 다 돌려봄 --> n(case bt case)개의 모델을 만듬
- 모델이기 때문에 fit()을 해야 최적의 파라미터를 찾음 --> case by case 수대로 fit()

```
sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring=None,refit=True, cv=None,
verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False, n_jobs=None )


- param_grid = fit_params --> dict 형태로 설정, hyper-parameter 튜닝한다
- refit = True : 가장 좋았던 parameter(Hyper-parameter) 조합으로 model 덮어쓴 후 점수를 냄  --> 오랜시간 걸림(나올 수 있는 case by case 다 돌려봄)
```

In [23]:
```python
skf= StratifiedKFold(n_splits=5,shuffle=True, random_state=11)
model=DecisionTreeClassifier(random_state=11)
mydic={"min_samples_split":[1,2,3],
       "min_samples_leaf":[1,2,3]}  # 손댈 자신 없으면 건들지 말기
models= GridSearchCV(model, scoring='accuracy', cv=skf, param_grid=mydic)

models.fit(X_train,y_train)
```

Out[23]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [24]:
```python
models.best_estimator_
```

Out[24]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(min_samples_leaf=3, min_samples_split=1, random_state=11)
```

In [25]:
```python
models.best_score_
```

Out[25]:
```
0.9666666666666668
```

In [26]:
```python
models.best_params_
```

Out[26]:
```
{'min_samples_leaf': 3, 'min_samples_split': 1}
```

In [27]:
```python
pred=models.predict(X_test)  # 공전 문제 --> pred 답안제출
accuracy_score(y_test,pred)
```

Out[27]:
```
0.9666666666666667
```

In [28]:
```python
ss=pd.DataFrame(models.cv_results_)
ss.head(3)
```

Out[28]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	param_min_samples_split	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	0.003589	0.001016	0.002195	0.000748	1	1	{'min_samples_leaf': 1, 'min_samples_split': 1}	0.875	0.958333	1.000000	0.958333	0.958333	0.950000	0.040825	4
1	0.002992	0.001262	0.001596	0.000488	1	2	{'min_samples_leaf': 1, 'min_samples_split': 2}	0.875	0.958333	1.000000	0.958333	0.958333	0.950000	0.040825	4
2	0.002193	0.000398	0.001618	0.000463	1	3	{'min_samples_leaf': 1, 'min_samples_split': 3}	0.875	0.958333	0.958333	0.958333	0.958333	0.941667	0.033333	6

K-Fold 외 교차검증(Cross Validation)

```
K-Fold : K개의 폴드로 나누어 cv횟수만큼 반복 학습

홀드아웃 (Holdout)    : 비복원추출
부트스트랩(Bootstrap) : 복원추출(63.2%)

LOOCV(Leave One Out Cross Validation)   : test 폴드(1개데이터)
LpOCV(Leave p     Out Cross Validation) : test 폴드(p개데이터)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: