Hyper parameter tuning

PinkLAB Edu

10 November, 2025

Table of Contents

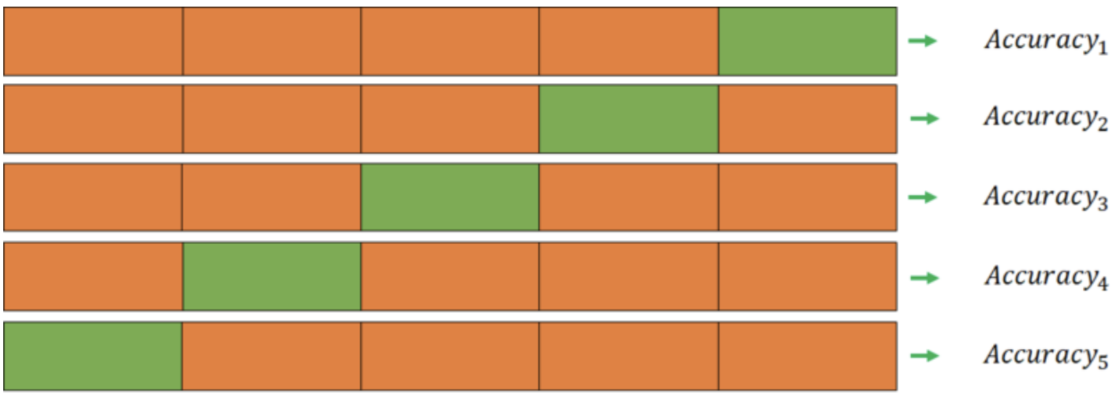
1 교차검증

1.1 교차검증

- 과적합 : 모델이 학습 데이터에만 과도하게 최적화된 현상.
  그로인해 일반화된 데이터에서는 예측 성능이 과하게 떨어지는 현상
- 지난번 와인 맛 평가에서 훈련용 데이터의 Acc는 72.94,
  테스트용 데이터는 Acc가 71.61%였는데, 누가 이 결과가 정말 괜찮은 것인지 묻는다면?
- 나에게 주어진 데이터에 적용한 모델의 성능을 정확히 표현하기 위해서도 유용하다

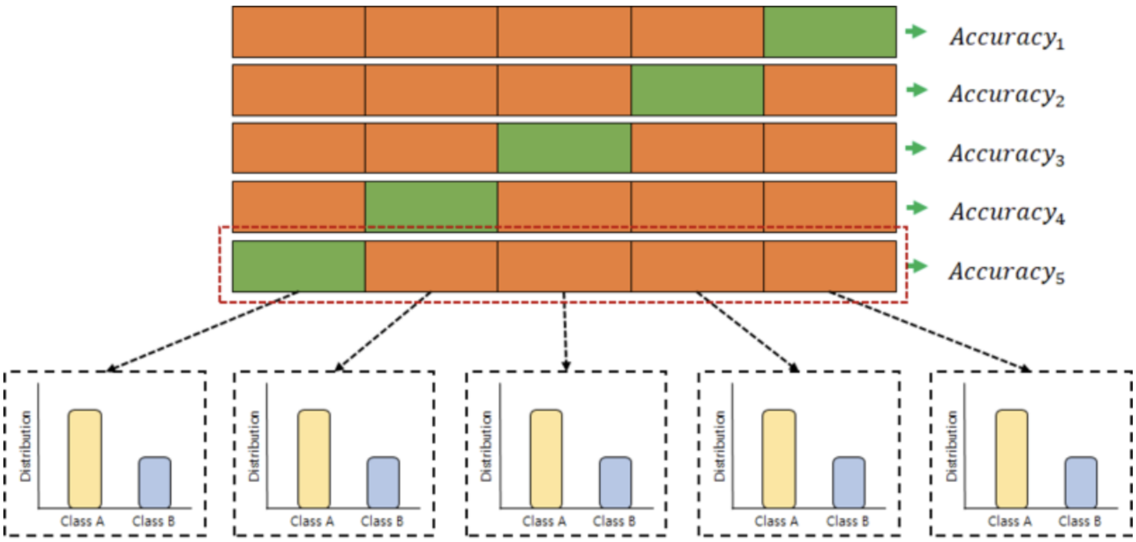
1.2 holdout

1.3 k-fold cross validation



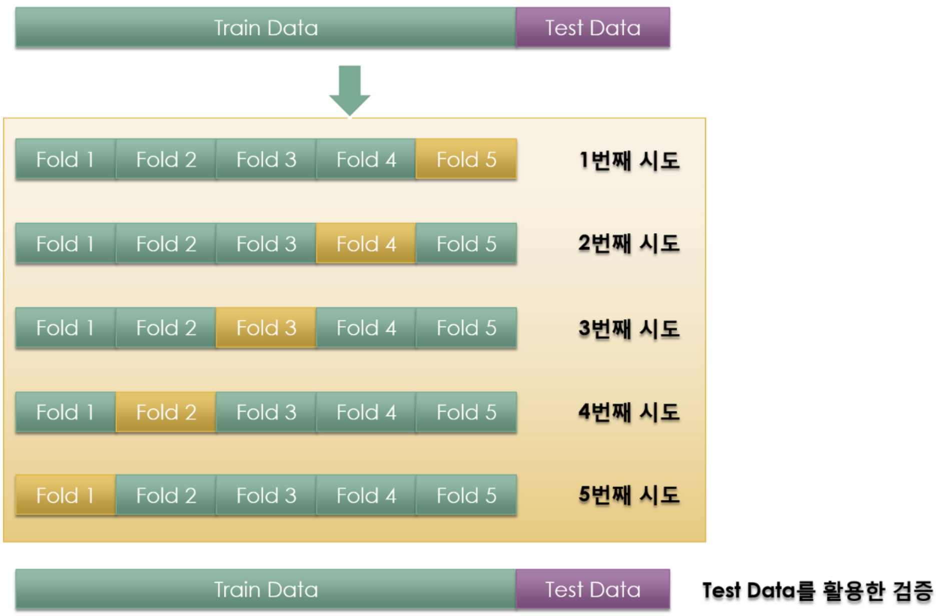
$$Accuracy = Average(Accuracy_1, \cdots, Accuracy_k)$$

1.4 stratified k-fold cross validation



$$Accuracy = Average(Accuracy_1, \cdots, Accuracy_k)$$

1.5 검증 validation이 끝난 후 test용 데이터로 최종 평가



2 교차 검증 구현하기

2.1 simple example

```python
import numpy as np
from sklearn.model_selection import KFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=2)

print(kf.get_n_splits(X))
print(kf)
print('='*30)

for train_idx, test_idx in kf.split(X):
    print("--- idx")
    print(train_idx, test_idx)
    print("--- train data")
    print(X[train_idx])
    print("--- val data")
    print(X[test_idx])
    print('='*30)
```
Python

2.2 결과

```
2
KFold(n_splits=2, random_state=None, shuffle=False)
==============================
--- idx
[2 3] [0 1]
--- train data
[[1 2]
 [3 4]]
--- val data
[[1 2]
 [3 4]]
==============================
--- idx
[0 1] [2 3]
--- train data
[[1 2]
 [3 4]]
--- val data
[[1 2]
 [3 4]]
==============================
```

2.3 다시 와인 맛 분류하던 데이터로

```python
import pandas as pd

red_url = "https://github.com/PinkWink/ML_tutorial/raw/refs/heads" + \
                                    "/master/dataset/winequality-red.csv"

white_url = "https://github.com/PinkWink/ML_tutorial/raw/refs/heads" +\
                                    "/master/dataset/winequality-white.csv"



red_wine = pd.read_csv(red_url, sep = ';')
white_wine = pd.read_csv(white_url, sep = ";")

red_wine['color'] = 1.
white_wine['color']= 0.

wine = pd.concat([red_wine, white_wine])
```
✓ 2.0s Python

2.4 와인 맛 분류기를 위한 데이터 정리

```python
wine['taste'] = [1. if grade > 5 else 0. for grade in wine['quality']]

X = wine.drop(['taste', 'quality'], axis = 1)
y = wine['taste']
```
✓ 0.0s Python

2.5 지난번 의사 결정 나무 모델로는?

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 13)
wine_tree = DecisionTreeClassifier(max_depth = 2, random_state = 13)
wine_tree.fit(X_train, y_train)

y_pred_tr = wine_tree.predict(X_train)
y_pred_test = wine_tree.predict(X_test)

print('Train Acc : ' , accuracy_score(y_train, y_pred_tr))
print("Test Acc : " , accuracy_score(y_test, y_pred_test))
```

✓ 0.4s Python

```
Train Acc :  0.7294593034442948
Test Acc :  0.7161538461538461
```

- 여기서 잠깐, 그러니까 누가, “데이터를 저렇게 분리하는 것이 최선인건가?”
- “저 acc를 어떻게 신뢰할 수 있는가?” 라고 묻는다면~

2.6 KFold

```python
from sklearn.model_selection import KFold

kfold = KFold(n_splits=5)
wine_tree_cv = DecisionTreeClassifier(max_depth=2, random_state=13)
```

✓ 0.0s Python

2.7 KFold는 index를 반환한다

```python
for train_idx, test_idx in kfold.split(X):
    print(len(train_idx), len(test_idx))
```

✓ 0.0s Python

```
5197 1300
5197 1300
5198 1299
5198 1299
5198 1299
```

2.8 각각의 fold에 대한 학습 후 acc

```python
cv_accuracy = []

for train_idx, test_idx in kfold.split(X, y):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    wine_tree_cv.fit(X_train, y_train)
    pred = wine_tree_cv.predict(X_test)
    cv_accuracy.append(accuracy_score(y_test, pred))

cv_accuracy
```
✓ 0.0s Python

```
[0.6007692307692307,
 0.6884615384615385,
 0.7090069284064665,
 0.7628945342571208,
 0.7867590454195535]
```

2.9 각 acc의 분산이 크지 않다면 평균을 대표 값으로 한다

```python
np.mean(cv_accuracy)
```
✓ 0.0s Python

```
np.float64(0.709578255462782)
```

2.10 StratifiedKFold

```python
from sklearn.model_selection import StratifiedKFold

skfold = StratifiedKFold(n_splits=5)
wine_tree_cv = DecisionTreeClassifier(max_depth=2, random_state=13)

cv_accuracy = []

for train_idx, test_idx in skfold.split(X, y):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    wine_tree_cv.fit(X_train, y_train)
    pred = wine_tree_cv.predict(X_test)
    cv_accuracy.append(accuracy_score(y_test, pred))

cv_accuracy
```

✓ 0.0s Python

```
[0.5523076923076923,
 0.6884615384615385,
 0.7143956889915319,
 0.7321016166281755,
 0.7567359507313318]
```

2.11 acc의 평균이 더 나쁘다

```python
np.mean(cv_accuracy)
```

✓ 0.0s Python

```
np.float64(0.6888004974240539)
```

- 이런 경우 어떻게 해야 할까?

2.12 cross validation을 보다 간편히

```python
from sklearn.model_selection import cross_val_score

skfold = StratifiedKFold(n_splits=5)
wine_tree_cv = DecisionTreeClassifier(max_depth=2, random_state=13)

cross_val_score(wine_tree_cv, X, y, scoring=None, cv=skfold)
```

✓ 0.0s Python

```
array([0.55230769, 0.68846154, 0.71439569, 0.73210162, 0.75673595])
```

2.13 depth가 높다고 무조건 acc가 좋아지는 것도 아니다

```
wine_tree_cv = DecisionTreeClassifier(max_depth=5, random_state=13)

cross_val_score(wine_tree_cv, X, y, scoring=None, cv=skfold)
```
✓ 0.0s Python

```
array([0.50076923, 0.62615385, 0.69745958, 0.7582756 , 0.74903772])
```

2.14 train score와 함께 보고 싶다면

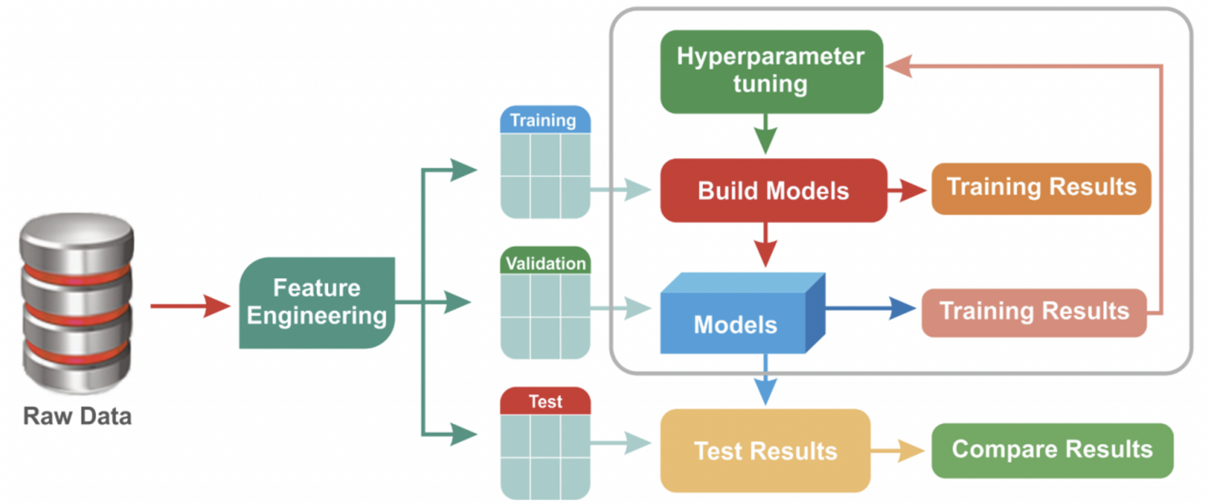
```
from sklearn.model_selection import cross_validate

cross_validate(wine_tree_cv, X, y, scoring=None, cv=skfold,
               return_train_score=True)
```
✓ 0.0s Python

```
{'fit_time': array([0.01190925, 0.01232815, 0.01158357, 0.01125073, 0.01109552]),
 'score_time': array([0.00132251, 0.00120187, 0.00128078, 0.00108957, 0.00108171]),
 'test_score': array([0.50076923, 0.62615385, 0.69745958, 0.7582756 , 0.74903772]),
 'train_score': array([0.78795459, 0.78045026, 0.77568295, 0.76356291,
0.76279338])}
```

- 현재 우리는 과적합 현상도 함께 목격하고 있다

3 하이퍼파라미터 튜닝

3.1 하이퍼파라미터 튜닝



- 모델의 성능을 확보하기 위해 조절하는 설정 값

3.2 튜닝 대상

- 결정나무에서 아직 우리가 튜닝해 볼만한 것은 max_depth이다.
- 간단하게 반복문으로 max_depth를 바꿔가며 테스트해볼 수 있을 것이다.
- 그런데 앞으로를 생각해서 보다 간편하고 유용한 방법을 생각해보자.

3.3 일단 다시 새파일에서 작업

```python
import pandas as pd

red_url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads"
    + "/master/dataset/winequality-red.csv"
)

white_url = (
    "https://github.com/PinkWink/ML_tutorial/raw/refs/heads"
    + "/master/dataset/winequality-white.csv"
)

red_wine = pd.read_csv(red_url, sep=";")
white_wine = pd.read_csv(white_url, sep=";")

red_wine["color"] = 1.0
white_wine["color"] = 0.0

wine = pd.concat([red_wine, white_wine])
wine["taste"] = [1.0 if grade > 5 else 0.0 for grade in wine["quality"]]

X = wine.drop(["taste", "quality"], axis=1)
y = wine["taste"]
```

Python

3.4 GridSearchCV

```python
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

params = {'max_depth' : [2, 4, 7, 10]}
wine_tree = DecisionTreeClassifier(max_depth = 2, random_state = 13)

gridsearch = GridSearchCV(estimator = wine_tree, param_grid = params , cv = 5)
gridsearch.fit(X, y)
```

✓ 0.3s Python

```
▸ GridSearchCV

▸ best_estimator_:
  DecisionTreeClassifier

  ▸ DecisionTreeClassifier
```

- 결과를 확인하고 싶은 파라미터를 정의하면 그만~

- cv는 cross validation

3.5 GridSearchCV의 결과

```python
import pprint

pp = pprint.PrettyPrinter(indent=4)
pp.pprint(gridsearch.cv_results_)
```

✓ 0.0s Python

```
{   'mean_fit_time': array([0.00581474, 0.00989447, 0.01649146, 0.022967  ]),
    'mean_score_time': array([0.00143118, 0.00139189, 0.00142927, 0.00143967]),
    'mean_test_score': array([0.6888005 , 0.66356523, 0.65340854, 0.64401587]),
    'param_max_depth': masked_array(data=[2, 4, 7, 10],
             mask=[False, False, False, False],
       fill_value=999999),
    'params': [   {'max_depth': 2},
                  {'max_depth': 4},
                  {'max_depth': 7},
                  {'max_depth': 10}],
    'rank_test_score': array([1, 2, 3, 4], dtype=int32),
    'split0_test_score': array([0.55230769, 0.51230769, 0.50846154, 0.51615385]),
    'split1_test_score': array([0.68846154, 0.63153846, 0.60307692, 0.60076923]),
    'split2_test_score': array([0.71439569, 0.72363356, 0.68360277, 0.66743649]),
    'split3_test_score': array([0.73210162, 0.73210162, 0.73672055, 0.71054657]),
    'split4_test_score': array([0.75673595, 0.7182448 , 0.73518091, 0.72517321]),
    'std_fit_time': array([0.00051776, 0.00040018, 0.00042915, 0.00078356]),
    'std_score_time': array([1.93331119e-04, 1.84420370e-04, 1.30381301e-04,
6.80574761e-05]),
    'std_test_score': array([0.07179934, 0.08390453, 0.08727223, 0.07717557])}
```

3.6 최적의 성능을 가진 모델은?

```
gridsearch.best_estimator_
```

✓ 0.0s Python

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=13)
```

```
gridsearch.best_score_
```

✓ 0.0s Python

```
np.float64(0.6888004974240539)
```

```
gridsearch.best_params_
```

✓ 0.0s Python

```
{'max_depth': 2}
```

3.7 만약 pipeline을 적용한 모델에 GridSearch를 적용하고 싶다면

```
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler


estimators = [('scaler', StandardScaler()),
              ('clf', DecisionTreeClassifier(random_state = 13))]
pipe = Pipeline(estimators)
```

✓ 0.0s Python

3.8 어렵지 않다

```python
param_grid = [ {'clf__max_depth' : [2, 4, 7, 10]}]

GridSearch = GridSearchCV(estimator = pipe, param_grid = param_grid, cv = 5)
GridSearch.fit(X, y)
```

✓ 0.4s Python

```
GridSearchCV

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                       ('clf',
                                        DecisionTreeClassifier(random_state=13))]),
             param_grid=[{'clf__max_depth': [2, 4, 7, 10]}])

best_estimator_: Pipeline

Pipeline(steps=[('scaler', StandardScaler()),
                ('clf', DecisionTreeClassifier(max_depth=2, random_state=13))])

StandardScaler

DecisionTreeClassifier
```

3.9 best 모델은?

```python
GridSearch.best_estimator_
```

✓ 0.0s Python

```
Pipeline

Pipeline(steps=[('scaler', StandardScaler()),
                ('clf', DecisionTreeClassifier(max_depth=2, random_state=13))])

StandardScaler
StandardScaler()

DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=13)
```

3.10 best_score_

```python
GridSearch.best_score_
```

✓ 0.0s Python

```
np.float64(0.6888004974240539)
```

```python
GridSearch.cv_results_
```

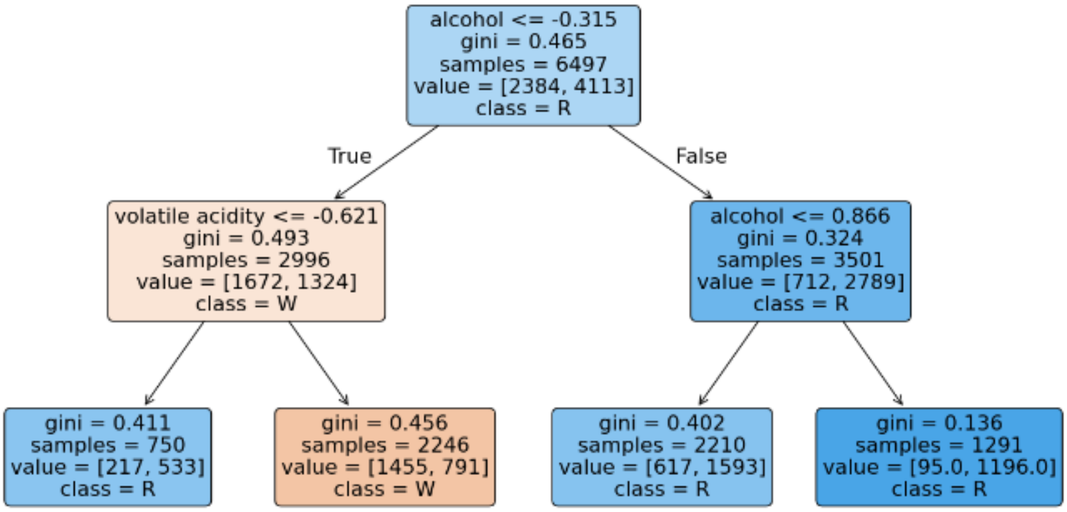
✓ 0.0s Python

```
{'mean_fit_time': array([0.00800552, 0.01074991, 0.01913009, 0.02879853]),
 'std_fit_time': array([0.00196962, 0.000345  , 0.00237366, 0.00145621]),
 'mean_score_time': array([0.00164223, 0.00155172, 0.00184488, 0.00205579]),
 'std_score_time': array([0.00029029, 0.00016769, 0.00039364, 0.0003351 ]),
 'param_clf__max_depth': masked_array(data=[2, 4, 7, 10],
              mask=[False, False, False, False],
        fill_value=999999),
 'params': [{'clf__max_depth': 2},
  {'clf__max_depth': 4},
  {'clf__max_depth': 7},
  {'clf__max_depth': 10}],
 'split0_test_score': array([0.55230769, 0.51230769, 0.50846154, 0.51615385]),
 'split1_test_score': array([0.68846154, 0.63153846, 0.60461538, 0.60230769]),
 'split2_test_score': array([0.71439569, 0.72363356, 0.68206313, 0.66589684]),
 'split3_test_score': array([0.73210162, 0.73210162, 0.73672055, 0.71054657]),
 'split4_test_score': array([0.75673595, 0.7182448 , 0.73518091, 0.72517321]),
 'mean_test_score': array([0.6888005 , 0.66356523, 0.6534083 , 0.64401563]),
 'std_test_score': array([0.07179934, 0.08390453, 0.08699322, 0.0769154 ]),
 'rank_test_score': array([1, 2, 3, 4], dtype=int32)}
```

3.11 Tree 확인해보기

```python
import matplotlib.pyplot as plt
from sklearn import tree

fig = plt.figure(figsize=(15, 8))
_ = tree.plot_tree(GridSearch.best_estimator_['clf'],
                   feature_names=X.columns,
                   class_names = ["W" , "R"],
                   rounded = True,
                   filled=True)
```

✓ 0.1s Python



3.12 잡기술 하나 - 표로 성능 결과를 정리하자

```python
import pandas as pd

score_df = pd.DataFrame(GridSearch.cv_results_)
score_df[['params' , 'rank_test_score', 'mean_test_score' , 'std_test_score']]
```

✓ 0.0s Python

	params	rank_test_score	mean_test_score	std_test_score
0	{'clf__max_depth': 2}	1	0.688800	0.071799
1	{'clf__max_depth': 4}	2	0.663565	0.083905
2	{'clf__max_depth': 7}	3	0.653408	0.086993
3	{'clf__max_depth': 10}	4	0.644016	0.076915

- accuracy의 평균과 표준편차를 보자~