Voting (1)

#01. 패키지 참조

```
from pandas import DataFrame, read_excel
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

#02. 데이터 가져오기

```
origin = read_excel('https://data.hossam.kr/G02/breast_cancer.xlsx')
origin.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	me symme
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

#03. 데이터 전처리

독립/종속 변수 분리

```
x = origin.drop('target', axis=1)
y = origin['target']
x.shape, y.shape
```

```
((569, 30), (569,))
```

데이터 표준화

```
scaler = StandardScaler()
std_x = scaler.fit_transform(x)
std_x[:5]
```

```
array([[ 1.09706398e+00, -2.07333501e+00,  1.26993369e+00,
         9.84374905e-01,  1.56846633e+00,  3.28351467e+00,
         2.65287398e+00,  2.53247522e+00,  2.21751501e+00,
         2.25574689e+00,  2.48973393e+00, -5.65265059e-01,
         2.83303087e+00,  2.48757756e+00, -2.14001647e-01,
         1.31686157e+00,  7.24026158e-01,  6.60819941e-01,
         1.14875667e+00,  9.07083081e-01,  1.88668963e+00,
        -1.35929347e+00,  2.30360062e+00,  2.00123749e+00,
         1.30768627e+00,  2.61666502e+00,  2.10952635e+00,
         2.29607613e+00,  2.75062224e+00,  1.93701461e+00],
       [ 1.82982061e+00, -3.53632408e-01,  1.68595471e+00,
         1.90870825e+00, -8.26962447e-01, -4.87071673e-01,
        -2.38458552e-02,  5.48144156e-01,  1.39236330e-03,
        -8.68652457e-01,  4.99254601e-01, -8.76243603e-01,
         2.63326966e-01,  7.42401948e-01, -6.05350847e-01,
        -6.92926270e-01, -4.40780058e-01,  2.60162067e-01,
        -8.05450380e-01, -9.94437403e-02,  1.80592744e+00,
        -3.69203222e-01,  1.53512599e+00,  1.89048899e+00,
        -3.75611957e-01, -4.30444219e-01, -1.46748968e-01,
         1.08708430e+00, -2.43889668e-01,  2.81189987e-01],
       [ 1.57988811e+00,  4.56186952e-01,  1.56650313e+00,
         1.55888363e+00,  9.42210440e-01,  1.05292554e+00,
         1.36347845e+00,  2.03723076e+00,  9.39684817e-01,
        -3.98007910e-01,  1.22867595e+00, -7.80083377e-01,
         8.50928301e-01,  1.18133606e+00, -2.97005012e-01,
         8.14973504e-01,  2.13076435e-01,  1.42482747e+00,
         2.37035535e-01,  2.93559404e-01,  1.51187025e+00,
        -2.39743838e-02,  1.34747521e+00,  1.45628455e+00,
         5.27407405e-01,  1.08293217e+00,  8.54973944e-01,
         1.95500035e+00,  1.15225500e+00,  2.01391209e-01],
       [-7.68909287e-01,  2.53732112e-01, -5.92687167e-01,
        -7.64463792e-01,  3.28355348e+00,  3.40290899e+00,
         1.91589718e+00,  1.45170736e+00,  2.86738293e+00,
         4.91091929e+00,  3.26373441e-01, -1.10409044e-01,
         2.86593405e-01, -2.88378148e-01,  6.89701660e-01,
         2.74428041e+00,  8.19518384e-01,  1.11500701e+00,
         4.73268037e+00,  2.04751088e+00, -2.81464464e-01,
         1.33984094e-01, -2.49939304e-01, -5.50021228e-01,
         3.39427470e+00,  3.89339743e+00,  1.98958826e+00,
         2.17578601e+00,  6.04604135e+00,  4.93501034e+00],
       [ 1.75029663e+00, -1.15181643e+00,  1.77657315e+00,
         1.82622928e+00,  2.80371830e-01,  5.39340452e-01,
         1.37101143e+00,  1.42849277e+00, -9.56046689e-03,
        -5.62449981e-01,  1.27054278e+00, -7.90243702e-01,
         1.27318941e+00,  1.19035676e+00,  1.48306716e+00,
        -4.85198799e-02,  8.28470780e-01,  1.14420474e+00,
        -3.61092272e-01,  4.99328134e-01,  1.29857524e+00,
        -1.46677038e+00,  1.33853946e+00,  1.22072425e+00,
         2.20556166e-01, -3.13394511e-01,  6.13178758e-01,
         7.29259257e-01, -8.68352984e-01, -3.97099619e-01]])
```

훈련/검증 데이터 분할

```
x_train, x_test, y_train, y_test = train_test_split(std_x, y, test_size=0.3, random_st
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((398, 30), (171, 30), (398,), (171,))
```

#04. Simple Model

개별 모델 정의

```
lr = LogisticRegression()
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier()
```

개별 모델을 앙상블 모델로 결합

```
vo = VotingClassifier(
    # 결합할 알고리즘의 리스트 (이름, 알고리즘객체) - 이름은 개발자가 마음대로 지정
    estimators=[('LR', lr), ('KNN', knn), ('DTREE', dt)],
    # hard: 다수결, soft: 확률의 평균
    voting='soft')

# 학습
vo.fit(x_train, y_train)

# 예측(검증데이터로...)
y_pred = vo.predict(x_test)

# 검증 데이터에 대한 정확도
score = accuracy_score(y_test, y_pred)
print(f'Voting 분류기 정확도: {score:.4f}')
```

```
Voting 분류기 정확도: 0.9708
```

개별 모델과의 결과 비교

```
classifiers = [lr, knn, dt]

for c in classifiers:
    c.fit(x_train, y_train)
    y_pred = c.predict(x_test)
    score = accuracy_score(y_test, y_pred)
    print(f'{c.__class__.__name__} 정확도: {score:.4f}')
```

```
LogisticRegression 정확도: 0.9766
KNeighborsClassifier 정확도: 0.9708
```

```
DecisionTreeClassifier 정확도: 0.9006
```

> 대부분의 예제들에서는 Voting의 성능이 좀 더 높게 나오지만 반드시 항상 그렇다고 할 수는 없다. 여러번 실행한다면 Voting 분류기의 성능이 더 높을 때도, 더 낮을 때도 발생하므로 여러번 테스트를 수행해야 한다.

#05. 최적 파라미터 찾기

```
vo = VotingClassifier(
    estimators=[('LR', LogisticRegression()),
                ('KNN', KNeighborsClassifier()),
                ('DTREE', DecisionTreeClassifier())],
    voting='soft')

# 테스트할 파라미터
params = {"voting": ['hard', 'soft'],
          'KNN__n_neighbors': [1, 3, 5, 7],
          'KNN__metric': ['euclidean', 'manhattan'],
          #'KNN__weights': ['uniform', 'distance'],
          'DTREE__max_depth': [3, 5, 7],
          #'DTREE__min_samples_split': [2, 3, 5],
          #'DTREE__min_samples_leaf': [1, 3, 5],
          'DTREE__max_features': [2, 3, 5]}

grid = GridSearchCV(vo, param_grid=params, cv=5)
grid.fit(x_train, y_train)

print(grid.best_params_)

result_df = DataFrame(grid.cv_results_['params'])
result_df['mean_test_score'] = grid.cv_results_['mean_test_score']
result_df.sort_values(by='mean_test_score', ascending=False)
```

```
{'DTREE__max_depth': 7, 'DTREE__max_features': 5, 'KNN__metric': 'euclidean', 'KNN__n_
```

	DTREE__max_depth	DTREE__max_features	KNN__metric	KNN__n_neighbors	voting	mean_tes
130	7	5	euclidean	3	hard	0.987405
34	3	5	euclidean	3	hard	0.987373
132	7	5	euclidean	5	hard	0.984905
80	5	5	euclidean	1	hard	0.984905
85	5	5	euclidean	5	soft	0.984905
...	...	...	...	...	...	...
53	5	2	euclidean	5	soft	0.964747
23	3	3	euclidean	7	soft	0.964684
119	7	3	euclidean	7	soft	0.962247
79	5	3	manhattan	7	soft	0.959747

	DTREE__max_depth	DTREE__max_features	KNN__metric	KNN__n_neighbors	voting	mean_tes
77	5	3	manhattan	5	soft	0.959715

144 rows × 6 columns

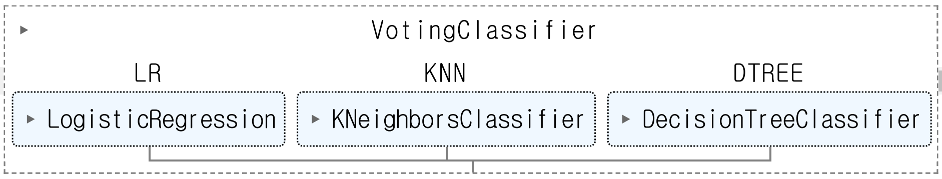
최적의 파라미터에 대한 학습 정확도

```
grid.best_score_
```

```
0.9874050632911393
```

최적의 파라미터를 갖는 객체

```
best = grid.best_estimator_
best
```



최적의 객체로 검증 데이터 예측

```
y_pred = best.predict(x_test)
y_pred[:5]
```

```
array([1, 1, 0, 1, 0], dtype=int64)
```

결과에 대한 정확도

```
score = accuracy_score(y_test, y_pred)
print(f'GridSearchCV 분류기 정확도: {score:.4f}')
```

```
GridSearchCV 분류기 정확도: 0.9766
```