Bagging (3)

1) sigleML함수를 사용하여 가장 높은 스코어를 보여주는 알고리즘 찾기

> KNeighborsClassifier

2) KNeighborsClassifier에 대하여 GridSearchGV를 적용하여 최적 파라미터 찾기

3) BaggingClassifier에 대하여 GridSearchGV를 적용.

> base_estimator=KNeighborsClassifier()에는 (2)번 단계에서 도출한 파라미터를 적용한다.

#01. 패키지 참조

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

from pandas import read_excel, DataFrame
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
```

#02. 분류 문제

1. 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	m 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

2. 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[:1]
```

```
array([[ 1.09706398, -2.07333501,  1.26993369,  0.9843749 ,  1.56846633,
         3.28351467,  2.65287398,  2.53247522,  2.21751501,  2.25574689,
         2.48973393, -0.56526506,  2.83303087,  2.48757756, -0.21400165,
         1.31686157,  0.72402616,  0.66081994,  1.14875667,  0.90708308,
         1.88668963, -1.35929347,  2.30360062,  2.00123749,  1.30768627,
         2.61666502,  2.10952635,  2.29607613,  2.75062224,  1.93701461]])
```

훈련/검증 데이터 분할

```
x_train, x_test, y_train, y_test = train_test_split(
    std_x, y, test_size=0.3, random_state=777)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

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

3. 분류 모델 구현

Bagging 모델 구현

```
clf = BaggingClassifier(
    base_estimator=KNeighborsClassifier(),
    random_state=777,
    n_jobs=-1)

params = {
    "bootstrap_features" : [True, False],
    "bootstrap": [True, False],
    "n_estimators": [30, 50]
}

grid = GridSearchCV(clf, 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)
```

```
{'bootstrap': True, 'bootstrap_features': True, 'n_estimators': 50}
```

	bootstrap	bootstrap_features	n_estimators	mean_test_score
1	True	True	50	0.964778
5	False	True	50	0.964778
3	True	False	50	0.964747
6	False	False	30	0.964747
7	False	False	50	0.964747
0	True	True	30	0.962278
4	False	True	30	0.962278
2	True	False	30	0.959715

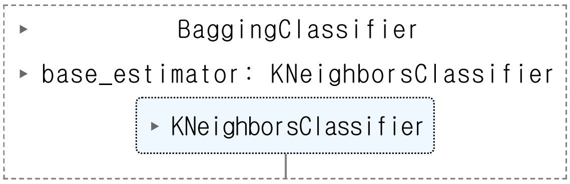
최적의 파라미터에 대한 학습 정확도

```
grid.best_score_
```

```
0.9647784810126583
```

최적의 파라미터를 갖는 객체

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



▸ BaggingClassifier

▸ base_estimator: KNeighborsClassifier

▸ KNeighborsClassifier

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
```