기계학습 결정 트리 정리노트

4조(이희구, 유제우)

※ 4조 의견

질문) 결정트리의 장점에는 무엇이 있을까?

- 결정트리 방식의 최대 장점은 데이터 전처리가 불필요하다는 것이다.

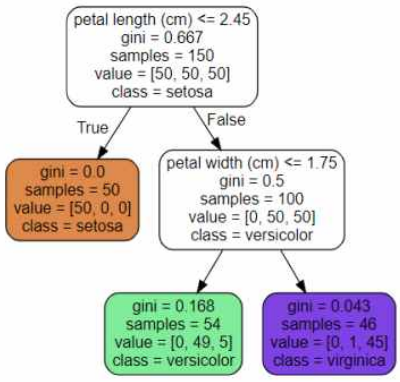
질문) 결정트리는 어떻게 구성되고 각 구성이 의미하는 것은 무엇일까?

- 결정트리는 Node, Root node, Leaf node로 구성되는데 Node는 가지치기가 시작되는 지점을 말하고 Root node는 맨 상단에 위치한 노드, Leaf node는 더 이상의 가지치기가 발생하지 않는 노드를 말한다.

질문) 결정트리는 깊이 조절 및 시각화는 어떻게 할까?

- 결정트리는 최대 깊이를 지정하여 가지치기의 깊이를 조절할 수 있다. 코드에서 결정트리의 깊이를 설정하기 위해서는 max_depth라는 값을 조정해주면 깊이가 조절된다. 이렇게 구성된 결정트리를 시각화하여 확인하기 위해서는 export_graphviz() 함수를 활용하면 된다.

질문) gini? 지니가 무엇일까? 여기에 해당하는 값들이 의미하는 것이 무엇일까?



▲ 그림 6-1 붓꽃 결정 트리

- gini는 해당 노드의 불순도 측정값을 의미한다. 모든 샘플이 동일 클래스에 속하면 불순도가 0이 되며 이것은 gini = 0을 의미한다.

- samples는 해당 노드 결정에 사용된 샘플 수이다.
- value는 해당 마디 결정에 사용된 샘플을 클래스별로 구분한 결과이다.
- class는 각 클래스별 비율을 계산하여 가장 높은 비율에 해당하는 클래스를 선정한다. 동일한 비율이면 낮은 인덱스를 선정한다.

질문) 결정트리로 확률 추정은 어떻게 하는 것일까?

- 결정트리는 한 샘플이 특정 클래스 k에 속할 확률을 추정할 수 있다. 계산된 클래스별 비율을 이용하여 새로운 샘플에 대한 예측을 실행한다.

```
tree_clf.predict_proba([[5, 1.5]])
array([[0.        , 0.90740741, 0.09259259]])

tree_clf.predict([[5, 1.5]])
array([1])
```

- Iris-Setosa = 0% (0/54), Iris-Versicolor = 90.7% (49/54), Iris-Virginica = 9.3% (5/54)
- 예측 : Iris-Versicolor 출력 (가장 높은 확률)

질문) CART 훈련? 그게 무엇일까?

탐욕적 알고리즘(greedy algorithm) 활용

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

여기에서 $\begin{cases} G_{\text{left/right}} \text{는 왼쪽/오른쪽 서브셋의 불순도} \\ m_{\text{left/right}} \text{는 왼쪽/오른쪽 서브셋의 샘플 수} \end{cases}$

▲ 식 6-2 분류에 대한 CART 비용 함수

- 탐욕적 알고리즘으로 여러 경우 중 하나를 결정해야 할 때마다 그 순간에 최적이라고 생각되는 것을 선택해 나가는 방식이다.
- 사이킷런은 결정 트리를 훈련시키기 위해 CART 알고리즘을 사용한다.
- 비용 함수를 최소화하는 특성 k와 해당 특성의 임계값 tk을 결정한다.

질문) CART 알고리즘을 활용하려면 k와 tk를 사용해야 하는데 고르는 방법은?

- J(k, tk)가 작을수록 불순도가 낮은 두 개의 부분집합으로 분할된다.
- 탐욕적 알고리즘은 해당 노드에 포함된 샘플을 지니 불순도가 가장 낮은, 즉, 가장 순수한

(pure) 두 개의 부분집합으로 분할한다.

- max_depth 깊이에 다다르거나 불순도를 줄이는 분할을 더 이상 찾을 수 없을 때, 또는 다른 규제의 한계에 다다를 때까지 반복한다.

질문) 결정트리의 계산 복잡도는 어때?

- 일반적으로 결정 트리는 거의 균형을 이루고 있으므로 결정 트리를 탐색하기 위해서는 약 O(log2(m)) 개의 노드를 거쳐야 한다. 각 노드는 하나의 특성값만 확인하기 때문에 예측에 필요한 전체 복잡도는 특성 수와 무관하게 O(log2(m))이다. 그러므로 큰 훈련 세트를 다룰 때도 예측 속도가 매우 빠르다.

질문) 지니 불순도와 엔트로피 중 어떤 것을 사용해야 할까?

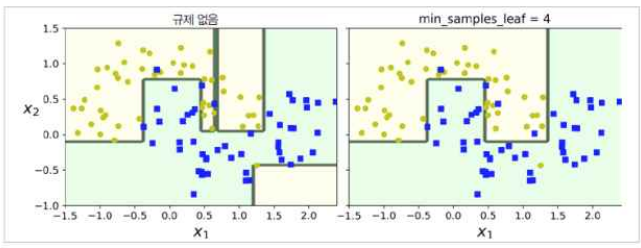
- 실제로는 큰 차이가 없이 둘 다 비슷한 트리를 만든다. 지니 불순도가 조금 더 계산이 빠르기 때문에 기본값으로 좋다. 다른 트리가 만들어지는 경우 지니 불순도가 가장 빈도 높은 클래스를 한쪽 가지(branch)로 고립시키는 경향이 있는 반면 엔트로피는 조금 더 균형잡힌 트리를 만든다.

질문) 사이킷런 DecisionTreeClassifier 규제하는 법은?

- max_depth로 결정 트리의 최대 높이 제한, min_samples_split로 노드를 분할하기 위해 필요한 최소 샘플 수 지정, min_samples_leaf로 리프 노드가 가지고 있어야 할 최소 샘플 수 지정, min_weight_fraction_leaf로 샘플 별로 가중치가 설정된 경우 가중치의 전체 합에서 해당 리프 노드에 포함된 샘플의 가중치의 합이 차지하는 비율 계산, max_leaf_nodes: 허용된 리프 노드의 최대 개수, max_features: 각 노드에서 분할 평가에 사용될 수 있는 최대 특성 수

- 규제를 높이는 방법 → min_ 접두사 사용 규제: 매개변수를 증가시킴킨다. max_ 접두사 사용 규제: 매개변수를 감소시킨다.

규제 매개변수 적용 전후 차이 비교 (왼편: 규제X / 오른편: 규제O)



▲ 그림 6-3 min_samples_leaf 매개변수를 사용한 규제

※ 오른편에 min_samples_leaf=4 적용

질문) 결정트리를 회귀에도 적용할 수 있나?

- 결정트리는 회귀 문제에도 사용할 수 있다. 사이킷런의 DecisionTreeRegressor를 사용하여 결정트리 회귀 모델을 사용할 수 있다.

- 잡음이 섞인 2차 함수 형태의 데이터셋에서 max_depth=2 설정

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)
```

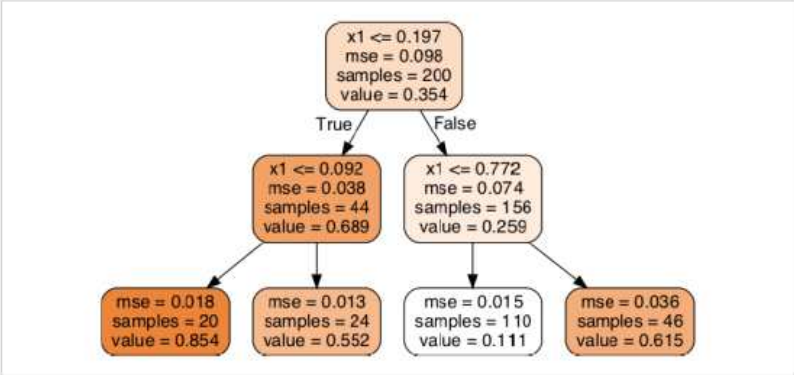


그림 6-4 회귀 결정 트리

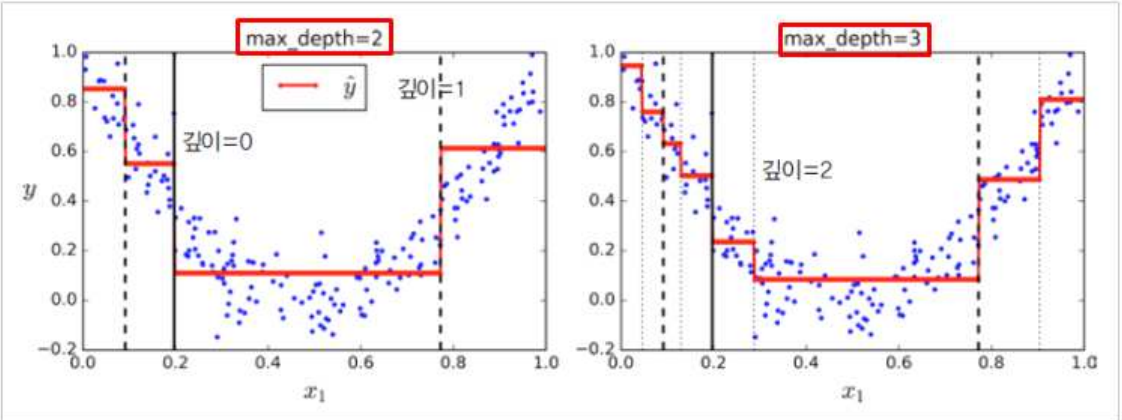
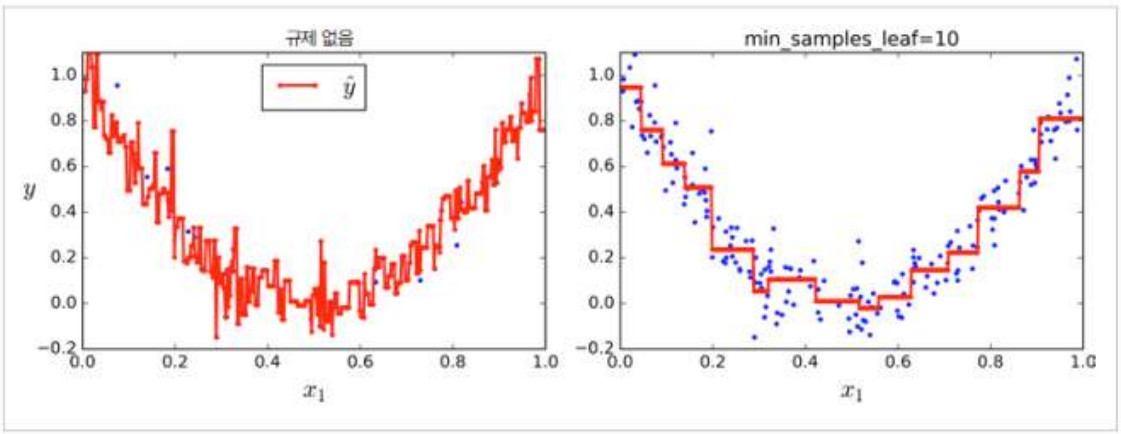


그림 6-5 두 개의 결정 트리 회귀 모델의 예측

질문) 결정트리 회귀모델에서 규제는 어떻게 할까?

- 분류에서와 같이 회귀 작업에서도 결정트리가 과대적합 되기 쉽다.

- 왼편: 규제가 없는 경우 (과대적합 발생) / 오른편: min_samples_leaf:10 (나름 괜찮음)

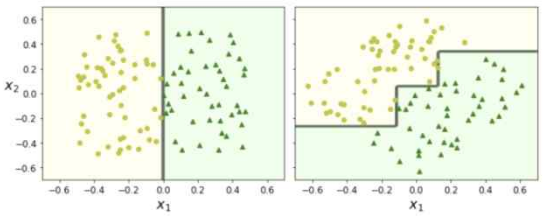


▲ 그림 6-6 결정 트리 회귀 모델의 규제

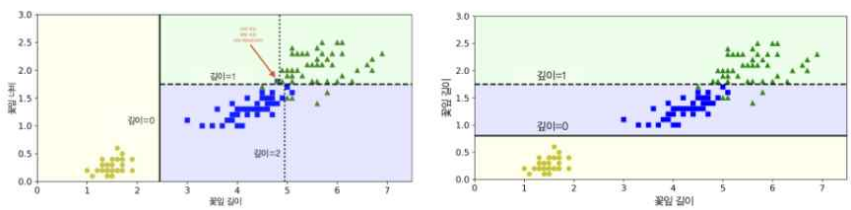
질문) 그럼 결정트리에 단점은 없을까?

- 결정트리는 여러 용도로 사용가능하고, 성능이 매우 우수하지만 훈련 세트에 민감하게 반응한다. 결정트리는 계단 모양의 결정 경계를 만든다.

- 단점1: 훈련 세트의 회전에 민감하다. 오른편 그래프는 왼편 그래프를 45도 회전시킨 훈련 세트를 학습시킨 것이다. 두 결정트리 모두 훈련 세트를 완벽하게 학습하지만 오른편쪽 모델은 일반화하기 어렵다.



- 단점2: 훈련 세트의 작은 변화에 매우 민감하다. 붓꽃 데이터에서 하나의 샘플을 제거한 후 학습시킬 때 매우 다르게 학습할 수 있다. 왼편 그래프는 모든 샘플 대상 훈련을 한 것이고, 오른쪽 그래프는 가장 넓은 Iris-Versicolor 샘플을 제거 후 훈련한 것이다.



4조 연습문제 7 풀이

저희 4조는 **max_depth값, 지니/엔트로피 파라미터 추가 및 모든 파라미터 범위 수정, StandardScaler 진행, noise값 조정** 방법을 사용하여 총 **4가지 테스트를 진행**하였고, 최종적으로 정확도 0.91을 도출해내었습니다.

Test_01

Test 1 - max_depth만 추가

```
1 from sklearn.datasets import make_moons
2
3 X, y = make_moons(n_samples=10000, noise=0.4, random_state=201804217)
```

```
[3] 1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=201804217)
```

```
[ ] 1 from sklearn.model_selection import GridSearchCV
2 from sklearn.tree import DecisionTreeClassifier
3
4 dt = DecisionTreeClassifier(random_state=201804217)
5
6 params = {'max_leaf_nodes': list(range(2, 300, 2)), 'min_samples_split': [2, 3, 4, 5, 6, 7], 'max_depth':list(range(1, 13))}
7 GridCV = GridSearchCV(dt, params)
8
9 GridCV.fit(X_train, y_train)
```

GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier

```
[ ] 1 GridCV.best_estimator_
```

DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, max_leaf_nodes=26, random_state=201804217)

```
[10] 1 from sklearn.metrics import accuracy_score
2
3 y_pred = grid_search.predict(X_test)
4 accuracy_score(y_test, y_pred)
```

0.855

- 10,000개의 샘플과 0.4의 노이즈를 가진 데이터셋을 생성한 후, 이를 80% 훈련 데이터와 20% 테스트 데이터로 나누었습니다. 훈련 데이터에는 Decision Tree 분류 모델을 생성하였고, 그리드 탐색(GridSearchCV)을 활용하여 Decision Tree의 하이퍼파라미터인 'max_leaf_nodes', 'min_samples_split', 'max_depth'와 같은 값을 최적화하였습니다. 최종 모델은 테스트 데이터에서 정확도를 측정하여 모델의 성능을 평가하였습니다.

Test_02

Test 2 - 지니, 엔트로피 파라미터 추가 및 모든 파라미터 범위 수정

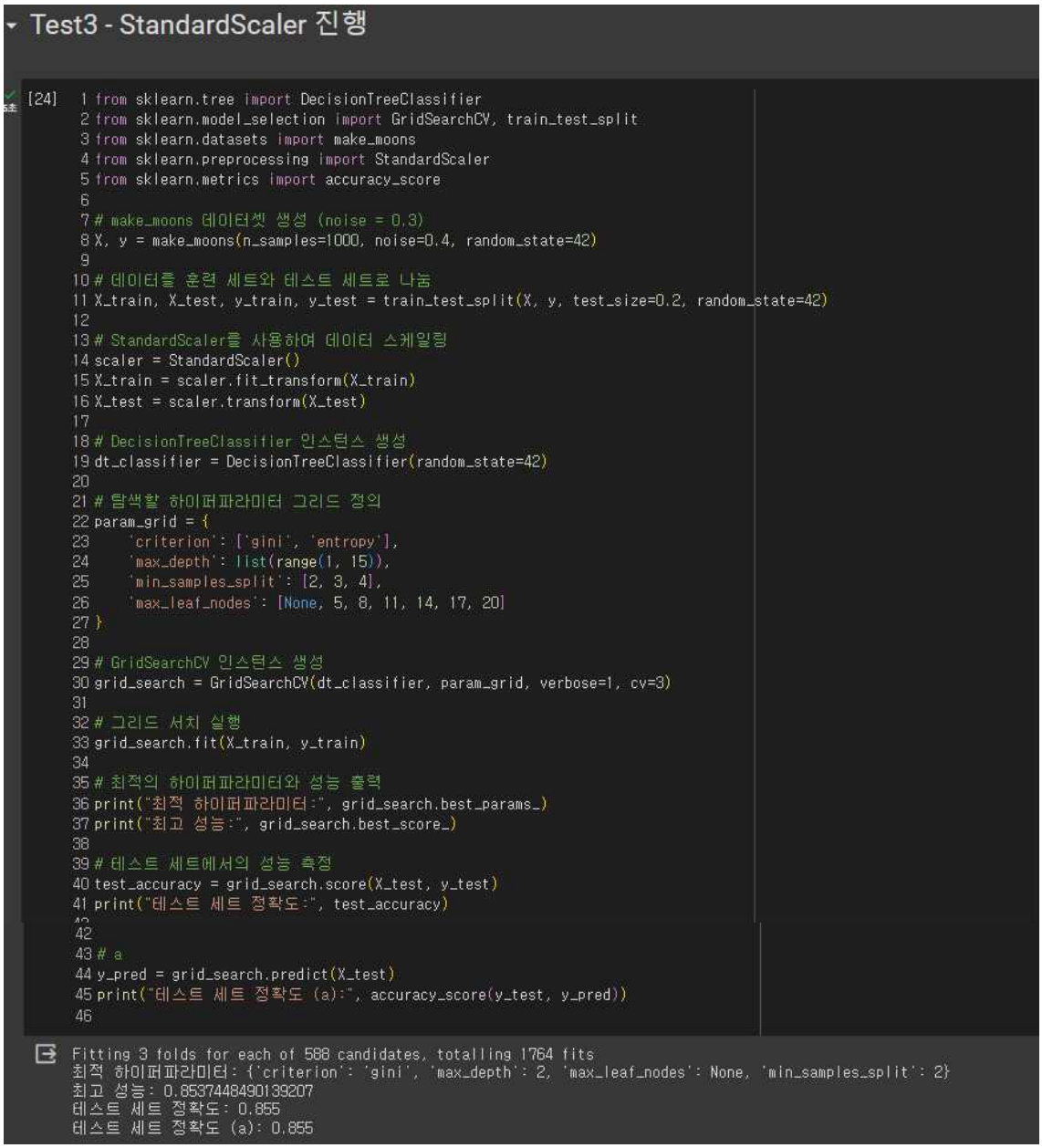
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=201804217)

params = {'max_leaf_nodes': list(range(2, 100, 2)), 'min_samples_split': [2, 3, 4, 5, 6, 7], 'max_depth':list(range(1, 13))}
GridCV = GridSearchCV(dt, params)

GridCV.fit(X_train, y_train)
```

```
# 필요한 라이브러리 import
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.datasets import make_moons

# make_moons 데이터셋 생성 (noise = 0.4)
X, y = make_moons(n_samples=1000, noise=0.4, random_state=42)

# 데이터를 훈련 세트와 테스트 세트로 나눔
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# DecisionTreeClassifier 인스턴스 생성
dt_classifier = DecisionTreeClassifier(random_state=42)

# 탐색할 하이퍼파라미터 그리드 정의
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': list(range(1,15)),
    'min_samples_split': [2, 3, 4],
    'max_leaf_nodes': [None, 5, 8, 11, 14, 17, 20]
}

# GridSearchCV 인스턴스 생성
grid_search = GridSearchCV(dt_classifier, param_grid, verbose=1, cv=3)

# 그리드 서치 실행
grid_search.fit(X_train, y_train)

# 최적의 하이퍼파라미터와 성능 출력
print("최적 하이퍼파라미터:", grid_search.best_params_)
print("최고 성능:", grid_search.best_score_)

# 테스트 세트에서의 성능 측정
test_accuracy = grid_search.score(X_test, y_test)
print("테스트 세트 정확도:", test_accuracy)

# a
y_pred = GridCV.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

```
Fitting 3 folds for each of 588 candidates, totalling 1764 fits
최적 하이퍼파라미터: {'criterion': 'gini', 'max_depth': 2, 'max_leaf_nodes': None, 'min_samples_split': 2}
최고 성능: 0.8537448490139207
테스트 세트 정확도: 0.855
0.845
```

- 이전 단계에서는 10,000개의 샘플과 0.4의 노이즈를 가진 데이터셋을 생성한 후, 80%의 훈련 데이터와 20%의 테스트 데이터로 분할하였습니다. 훈련 데이터에는 Decision Tree 분류 모델을 생성하였고, 그리드 탐색(GridSearchCV)을 활용하여 Decision Tree의 하이퍼파라미터를 최적화하였습니다. 이번 단계에서는 **지니와 엔트로피를 사용하여 모델의 불순도 척도를 변경**하였고, **'max_leaf_nodes', 'min_samples_split', 'max_depth'와 같은 모든 하이퍼파라미터의 범위를 수정**하였습니다. 최종 모델은 테스트 데이터에서 정확도를 측정하여 모델의 성능을 평가하였습니다.

Test_03



- 데이터셋은 1,000개의 샘플로 구성되고, noise가 0.4로 설정된 'make_moons' 데이터셋을 생성하였습니다. 이 데이터는 훈련 데이터와 테스트 데이터로 8:2 비율로 분할하였으며, 데이터 스케일링을 위해 '**StandardScaler**'를 사용하였습니다. Decision Tree 분류 모델을 사용하였고, 'criterion', 'max_depth', 'min_samples_split', 'max_leaf_nodes' 등의 하이퍼파라미터를 그리드 탐색(GridSearchCV)을 통해 조정하였습니다. 최적의 하이퍼파라미터와 성능 지표를 출력하고, 테스트 데이터에서 모델의 정확도를 측정하여 모델의 성능을 평가하였습니다.

Test_04

Test4 - make_moons함수로 데이터셋 생성시 noise 값이 작을 수록 모델 성능을 올리는 패턴을 확인함. 따라서 noise를 0.3으로 감소시켜 진행

```
[20]  1 # 필요한 라이브러리 import
      2 from sklearn.tree import DecisionTreeClassifier
      3 from sklearn.model_selection import GridSearchCV, train_test_split
      4 from sklearn.datasets import make_moons
      5
      6 # make_moons 데이터셋 생성 (noise = 0.3)
      7 X, y = make_moons(n_samples=1000, noise=0.3, random_state=42)
      8
      9 # 데이터를 훈련 세트와 테스트 세트로 나눔
     10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     11
     12 # DecisionTreeClassifier 인스턴스 생성
     13 dt_classifier = DecisionTreeClassifier(random_state=42)
     14
     15 # 탐색할 하이퍼파라미터 그리드 정의
     16 param_grid = {
     17     'criterion': ['gini', 'entropy'],
     18     'max_depth': list(range(1,15)),
     19     'min_samples_split': [2, 3, 4],
     20     'max_leaf_nodes': [None, 5, 8, 11, 14, 17, 20]
     21 }
     22
     23 # GridSearchCV 인스턴스 생성
     24 grid_search = GridSearchCV(dt_classifier, param_grid, verbose=1, cv=3)
     25
     26 # 그리드 서치 실행
     27 grid_search.fit(X_train, y_train)
     28
     29 # 최적의 하이퍼파라미터와 성능 출력
     30 print("최적 하이퍼파라미터:", grid_search.best_params_)
     31 print("최고 성능:", grid_search.best_score_)
     32
     33 # 테스트 세트에서의 성능 측정
     34 test_accuracy = grid_search.score(X_test, y_test)
     35 print("테스트 세트 정확도:", test_accuracy)
     36
     37 # a
     38 y_pred = grid_search.predict(X_test)
     39 print(accuracy_score(y_test, y_pred))
     40
     41

Fitting 3 folds for each of 588 candidates, totalling 1764 fits
최적 하이퍼파라미터: {'criterion': 'gini', 'max_depth': 10, 'max_leaf_nodes': 20, 'min_samples_split': 4}
최고 성능: 0.9037528277622896
테스트 세트 정확도: 0.91
0.91
```

- 데이터셋은 1,000개의 샘플로 구성되고, **noise가 0.3**로 설정된 'make_moons' 데이터셋을 생성하였습니다. 이 데이터는 훈련 데이터와 테스트 데이터로 8:2 비율로 분할하였고, Decision Tree 분류 모델을 사용하였습니다. 그리드 탐색(GridSearchCV)을 통해 'criterion', 'max_depth', 'min_samples_split', 'max_leaf_nodes' 등의 하이퍼파라미터를 탐색하고, 최적의 하이퍼파라미터와 성능을 출력하였습니다. 마지막으로 테스트 데이터를 사용하여 모델의 정확도를 측정하였습니다.