Ch06_결정 트리_0731

결정 트리(decision tree)

결정 트리는 분류와, 회귀 작업, 다중출력 작업이 기능하며, 복합한 데이터셋을 학습할 수 있는 머신러닝 알고리즘이다. 또힌 랜덤 포레스트의 기본구성 요소이기도 하다.

- 결정 트리의 훈련，시각화 예측 방법
- 사이킷런의 CART 훈련 알고리즘
- 트리에 규제를 가하는 방법
- 회귀 문제에 적용하는 방법
- 결정 트리의 제약 사항

▼ 6.1 결정 트리 학습과 시각화

- ex) 붓꽃 데이터셋에 DecisionTreeClassifier를 훈련

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:,2:] #꽃잎의 길이와 너비
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X,y)
```

```
#<그림 6-1. 붓꽃 결정 트리> 생성 코드

from graphviz import Source
from sklearn.tree import export_graphviz

export_graphviz(
    tree_clf,
```

```
    out_file = os.path.join(IMAGES_PATH, "iris_tree.dot"),
    feature_names = iris.feature_names[2:],
    class_names=iris.target_names,
    rounded=True,
    filled=True
)

Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```

→ `export_grahpviz()`

: 그래프 정의를 iris_tree.dot 파일로 출력하여 훈련된 결정 트리를 시각화 할 수 있다.

→ 이 .dot 파일을 Graphviz 패키지에 있는 dot 명령줄 도구로 PDF나 PNG같은 포맷으로 변경한다.

→ `$dot - Tpng iris_tree.dot 0 -0 iris_tree.png` : .dot 파일을 .png 파일로 변경

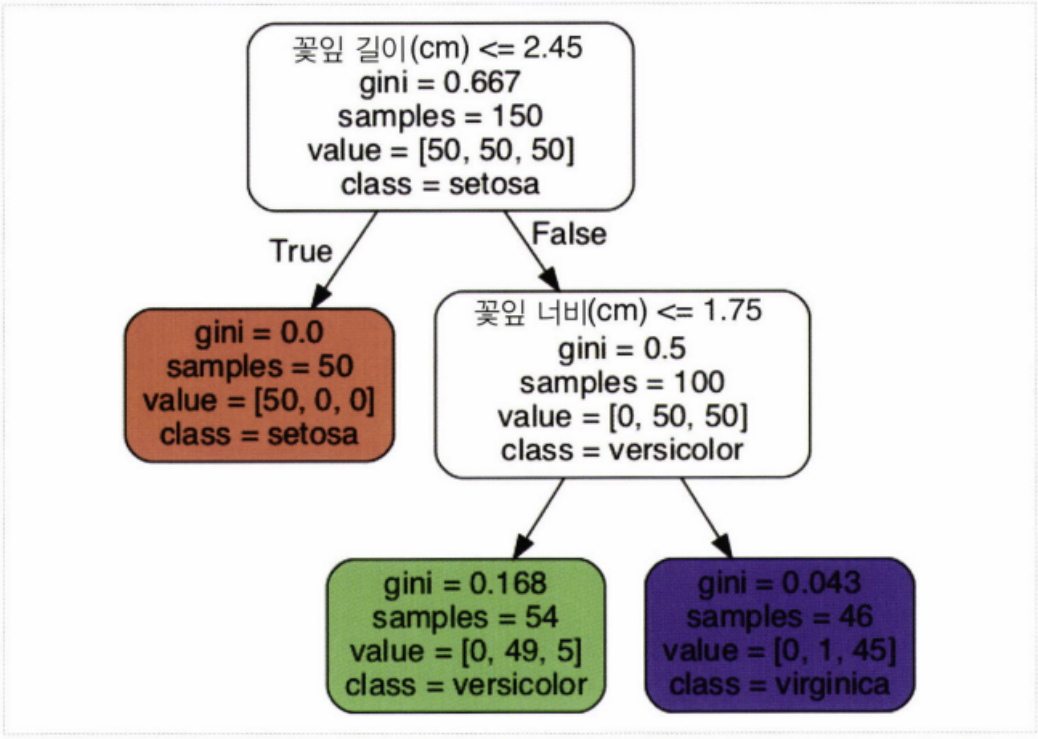


그림 6-1 붓꽃 결정 트리

▼ 6.2 예측하기

- **ex. 그림 6-1**

새로 발견한 붓꽃의 품종을 분류하려 한다고 가정하자.

1) 먼저 루트 노드(root node, 깊이가 0인 맨 꼭대기의 노드) 에서 시작한다. 이 노드는 꽃잎의 길이가 2.45cm보다 짧은지 검사한다,

1-1) 꽃잎의 길이 < 2.45cm

: 루트 노드에서 왼쪽의 자식 노드(child node) (깊이 1, 왼쪽노드)로 이동한다. 이 경우 이 노드가 리프노드(leaf node, 자식 노드를 가지지 않는 노드) 이므로 추가적인 검사를 하지 않는다. 그냥 노드에 있는 측 클래스를 보고 결정 트리가 새로 발견한 꽃의 품종을 Iris-Setosa (class=setosa) 라고 예측한다.

1-2) 꽃잎의 길이 > 2.45 cm

: 루트 노드의 오른쪽 자식 노드로 이동한다. 이 노드는 리프 노드가 아니라서 추가로 '꽃잎의 너비가 1.75cm 보다 작은지 검사해야 한다.

2) (1-2) 단계에서 만약 꽃잎의 너비가 1.75cm보다 작다면 Iris-Versicolor(깊이 2, 왼쪽)으로, 크다면 Iris-Virginical(깊이 2, 오른쪽)으로 분류된다.

- **결정 트리의 장점**

: 데이터 전처리(특성의 스케일을 맞추거나 평균을 원점에 맞추는 일)가 거의 필요하지 않다.

노드의 속성

- **sample 속성**

: 열마나 많은 훈련 이 적용되었는지 헤아린 것이다.

ex) 100개의 훈련 샘플의 꽃잎 길이가 2.45cm 보다 길고(깊이 1, 오른쪽) , 그 중 54개 샘플의 꽃잎 너비가 1.75 보다 짧다(깊이 2, 왼쪽)

- **value 속성**

: 각 클래스에 얼마나 많은 훈련 샘플이 있는지 알려준다.

ex) 맨 오른쪽 아래 노드는 Iris-Setosa가 0개, Iris-Versicolor가 1개, Iris -Virginica가 54개 있다.

- **gini 속성**

: 불순도를 측정한다. 한 노드의 모든 샘플이 같은 클래스에 속해 있다면, 이 노드를 순수(gini=0)하다고 표현한다.

ex) 깊이 1의 왼쪽 노드는 Iris-Setosa 훈련 샘플만 가지고 있으므로 순수 노드이고 gini 점수 =0 이다.

아래의 식은 훈련 알고리즘이 i번째 노드의 지니점수 G_i를 계산하는 방법이다.

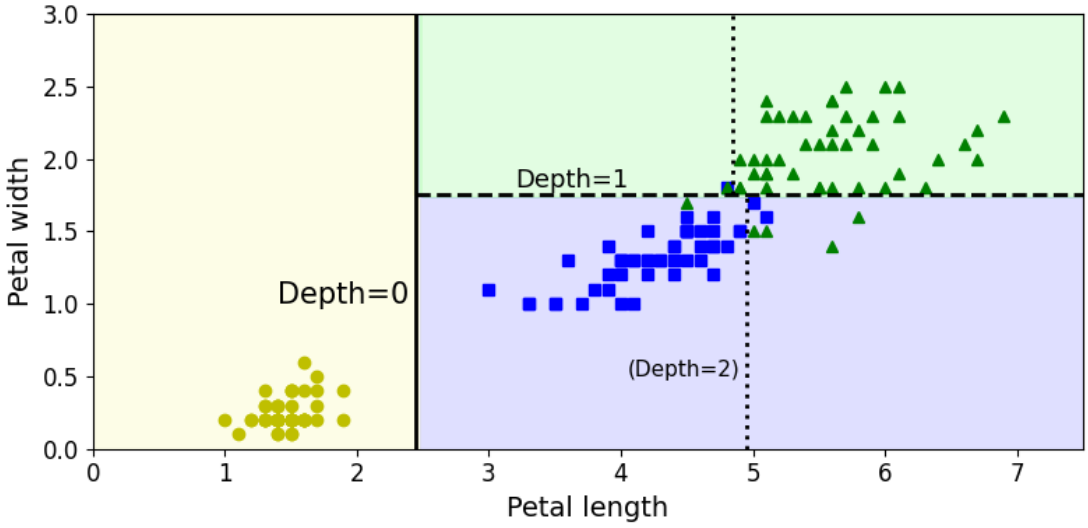
식 6-1 지니 불순도

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

→ $p_{i,k}$는 i번째 노드에 있는 훈련 샘플 중 클래스 k에 속한 샘플의 비율이다.

ex) (깊이 2, 왼쪽) 노드의 gini 점수 = $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$

- **결정 트리의 결정 경계**



→ 굵은 수직선이 루트노드(Depth=0)의 결정 경계(꽃잎 =2 .4 5cm)를 나타낸다.

→ 왼쪽 영역은 순수 노드 (Iris-Setosa만 있음)이기 때문에 더는 나눌 수 없다.

→ 오른쪽 영역은 순수노드가 아니므로 Depth=1의 오른쪽 너드는 꽃잎 너비 1.75cm 에서 나누어진다(파선).

→ max_ depth=2 로 설정했기 때문에 결정 트리는 더 분할되지 않는다. max_depth=3으로 하면 Depth=2의 두 노드가 각각 결정 경계를 추가로 만든다(점선).

- **모델 해석** : 화이트박스(white box)와 블랙박스(black box)

  - 화이트박스(white box)

  : 직관적이고 결정 방식을 이해하기 쉬운 모델

  ex)결정 트리

  - 블랙박스(black box)

  : 성능이 뛰어나고 예측을 만드는 연산 과정을 쉽게 확인할 수 있다. 하지만 왜 그런 예측을 만는지는 설명하기 어렵다.

  ex) 랜덤 포레스트, 신경망
  : 신경망이 어떤 사람이 사진에 있다고 판단했을 때 무엇이 이런 예측을 낳게 했는지 파악하기 어렵다. 반면에 결정 트리는 필요하다면 수동으로 직접 따라 해볼 수도 있는 간단하고 명확한 분류 방법을 사용한다.

▼ 6.3 클래스 확률 측정

결정 트리는 한 샘플이 특정 클래스 k에 속할 확률을 추정할 수 있다.

먼저 이 샘플에 대해 리프 노드를 찾기 위해 트리를 탐색하고 그 노드에 있는 클래스 k의 훈련 샘플의 비율을 반환한다.

ex) 길이가 5cm이고 너비가 1.5cm인 꽃잎을 발견했다고 가정한다. 이에 해당하는 리프 노드는 Depth=2에서 왼쪽 노드이므로 결정 트리는 그에 해당하는 확률을 출력한다. 즉, Iris-Setosa는 0%(0/54), Iris-Versicolor는 90.7%(49/ 54), IrisVirginica는 9.3%(5/ 54) 이다. 만약 클래스를 하나 예측한다면 가장 높은 확률을 가진 IrisVersicolor (클래스 1)을 출력한다.

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

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

→ 추정된 확률은 [그림 6-2] 의 오른쪽 아래 사각형 안에서는 어느 위치든 동일하다.
ex) 길이가 6cm, 너비가 1.5cm인 꽃잎도 획률이 같다.

▼ 6.4 CART 훈련 알고리즘

사이킷런은 결정 트리를 훈련시키기 위해 (즉, 트리를 성장 키기 위해) CART(classificaltion and regression tree) 알고리즘을 사용한다. 먼저 훈련 세트를 하나의 특성 k의 임곗값 t_k를 사용해 두개의 서브셋으로 나눈다. (예를 들면 꽃잎의 길이 $\leq 2.45cm$)

이때, k와 t_k는 (크기에 떠른 가중치가 적용된) 가장 순수한 서브셋으로 나눌 수 있는 (k, t_k)짝을 찾는다.

이 알고리즘이 최소화해야 하는 비용 함수는 다음과 같다.

식 6-2 분류에 대한 CART 비용 함수

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

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

CART 알고리즘이 훈련 세트를 성공적으로 둘로 나누었다면, 같은 방식으로 서브셋을 또 나누고, 다음엔 서브셋의 서브셋을 나누고 이런 식으로 계속 반복한다. 이 과정은 (max_depth 매개변수로 정의된) 최대 깊이가 되면 중지하거나 불순도를 줄이는 분할을 찾을 수 없을 때 멈추게 된다.

다른 몇 개의 매개변수도 중지 조건에 관여한다.

→ `min _samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf`, `max_leaf_nodes`

- **cuation**

: CART 알고리즘은 탐욕적 알고리즘(greedy algorithm)이다. 맨 위 루트노드에서 최적의 분할을 찾으며, 이어지는 각 단계에서 이 과정을 반복한다. 현재 단계의 분할이 몇 단계를 거쳐 가장 낮은 불순도로 이어질 수 있을지 없을지는 고려하지 않는다. 탐욕적 알고리즘은 종종 납득할 만한 훌륭한 솔루션을 만들어내지만, 최적의 솔루션을 보장하지는 않는다.

▼ 6.5 계산 복잡도

예측을 하려면 결정 트리를 루트 노드에서부터 리프 노드까지 탐색해야 한다. 일반적으로 결정 트리는 거의 균형을 이루고 있으므로, 결정 트리를 탐색하기 위해서는 약 $O(log_2(m))$ 개의 노드를 거쳐야 한다. 각 노드는 하나의 특성값만 확인하기 때문에, 예측에 필요한 전체 복잡도는 특성 수와 무관하게 $O(log_2(m))$이다. 그래서 큰 훈련 세트를 다룰 때도 예측 속도가 매우 빠르다.

훈련 알고리즘은 각 노드에서 모든 훈련 샘플의 모든(또는 max features가 지정되었다면 그보다는 적은) 특성을 비교한다. 각 노드에서 모든 샘플의 모든 특성을 비교하면 훈련 복잡도는 $O(n * mlog(m))$이 된다.

훈련 세트가(수천 개 이하의 샘플 정도로) 작을 경우, 사이킷런은 `presort=True`로 지정하면 미리 데이터를 정렬하여 훈련 속도를 높일 수 있다. 하지만 훈련 세트가 클 경우에는 속도가 많이 느려진다.

▼ 6.6 지니 불순도 또는 엔트로피

> ## 엔트로피 불순도

기본적으로 지니 불순도가 사용되지만 `criterion` 매개변수를 `"entropy"` 로 지정하여 엔트로피 불순도를 시용할 수 있다. 어떤 세트가 한 클래스의 샘플만 담고 있다면 엔트로피가 0이다.

i번째 노드의 엔트로피 정의는 다음과 같다.

식 6-3 엔트로피

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2(p_{i,k})$$

→ ex) [그림 6-1]에서 Depth=2의 왼쪽 노드의 엔트로피는 $-(49/54) - 5/54log_2(5/54) \approx 0.445$이다.

- **엔트로피 불순도 vs 지니 불순도**

: 실제로는 큰 차이가 없으며, 비슷한 트리를 만들어낸다. 지니 불순도가 조금더 계산이 빠르기 때문에 기본값으로 좋다. 그러나 다른 트리가 만어지는 경우, 지니 불순도가 가장 빈도 높은 클래스를 한쪽 가지로 고립시커는 경향이 있는 반면, 엔트로피는 조금 더 균형 잡힌 트리를 만든다.

▼ 6.7 규제 매개변수

결정 트리는 훈련 데이터에 대한 제약 사항이 거의 없다. (선형 모텔은 데이터가 선형일 것이라고 가정). 제한을 두지 않으면 트리가훈련 데이터에 아주 가깝게 맞추려고 해서 대부분 과대적합되기 쉽다. 결정 트라는 모델 파라미터가 전혀 없는 것이 아니라, 훈련되기 전에 피라미터 수가 결정되지 않기 때문에 이런 모델을 **비파라미터 모델(nonparameter model)**이라고 한다. 그래서 모델 구조가 데이터에 맞춰져서 고정되지 않고 자유롭다.

파라미터 모델(paramter model) 미리 정의된 델 파라미터 수를 가지므로 자유도가 제한되고 과대적합될 위험이 줄어든다. 하지만 과소적합 될 위험은 커진다.

훈련 데이터에 대한 과대적합을 피하기 위해 학습할 때 결정 트리의 자유도를 제한할 필요가 있다. 이를 규제라고 한다. 규제 매개변수는 사용하는 알고리즘에 따라 다르지만, 보통 적어도 결정 트리의 최대 깊이는 제어할 수 있다. 사이킷런에서는 `max_depth`매개변수로 이를 조절한다. 기본값은 제한이 없는 것 을 의미하는 `None`이다). `max_depth`를 줄이면 모델을 규제하게 되고 과대적합의 위험이 감소한다.

`DecisionTreeClassifier` 에는 결정 트리의 형태를 제한히는 매개변수가 몇 개 있다.

→ `min samples_split`(분할 되기 위해 노드가 가져야 하는 최소 샘플 수). `min_samp les_leaf` (리프 노드가 가지고 있어야 할 최소 샘플 수). `min_weight_fraction_leaf` (`min samples leaf` 와 같지만 기중치가 부여된 전체 샘플 수에서의 비율), `max_leaf nodes` *(리프 노드의 최대 수)*, `max features` (각 노드에서 분할에 사용할 특성의 최대 수)

`min_`으로 시작하는 매개변수를 증가 시키거나 `max_`로 시작하는 매개변수를 감소시키면 모델에 규제가 커진다.

- **결정트리 : moons dataset**

```
from sklearn.datasets import make_moons
Xm, ym = make_moons(n_samples=100, noise=0.25, random_state=53)

deep_tree_clf1 = DecisionTreeClassifier(random_state=42)
deep_tree_clf2 = DecisionTreeClassifier(min_samples_leaf=4, random_state=42)
deep_tree_clf1.fit(Xm, ym)
deep_tree_clf2.fit(Xm, ym)

fig, axes = plt.subplots(ncols=2, figsize=(10, 4), sharey=True)
plt.sca(axes[0])
plot_decision_boundary(deep_tree_clf1, Xm, ym, axes=[-1.5, 2.4, -1, 1.5], iris=False)
plt.title("No restrictions", fontsize=16)
plt.sca(axes[1])
plot_decision_boundary(deep_tree_clf2, Xm, ym, axes=[-1.5, 2.4, -1, 1.5], iris=False)
plt.title("min_samples_leaf = {}".format(deep_tree_clf2.min_samples_leaf), fontsize=14)
plt.ylabel("")

save_fig("min_samples_leaf_plot")
plt.show()
```

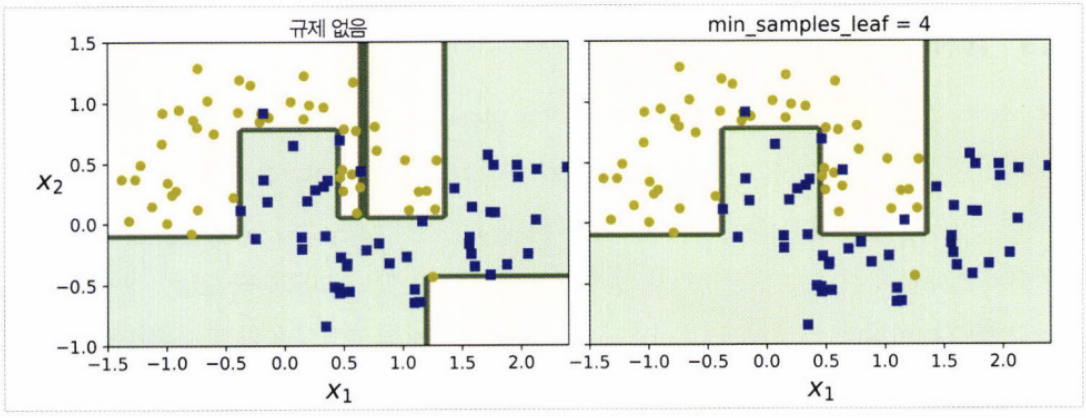


그림 6-3 min_samples_leaf 매개변수를 사용한 규제

→ moons 데이터셋에 훈련 시킨 두 개의 결정 트리를 보여준다.

→ 왼쪽 결정 트리

: 기본 매개변수를 사용하여 훈련(즉，규제가 없다) ,

→ 오른쪽 결정 트리

: `min_samples leaf=4`로 하여 훈련

→ 왼쪽 모델은 확실히 과대적합 되었고, 오른쪽 모델은일반화 성능이 더 좋을 것 같아 보인다.

▼ 6..8 회귀

결정트리는 회귀 문제에도 사용할 수 있다.

ex) 사이킷런의 `DecisionTreeRegressor`를 사용해 잡음이 섞인 2차 힘수 형태의 데이터셋에서 `max_depth=2` 설정으로 회귀 트리를 만든다.

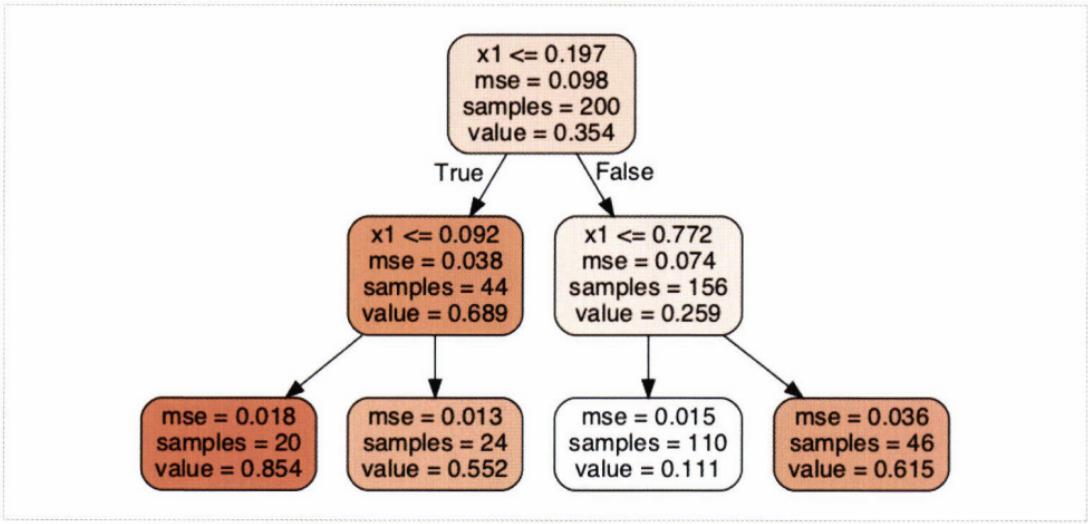


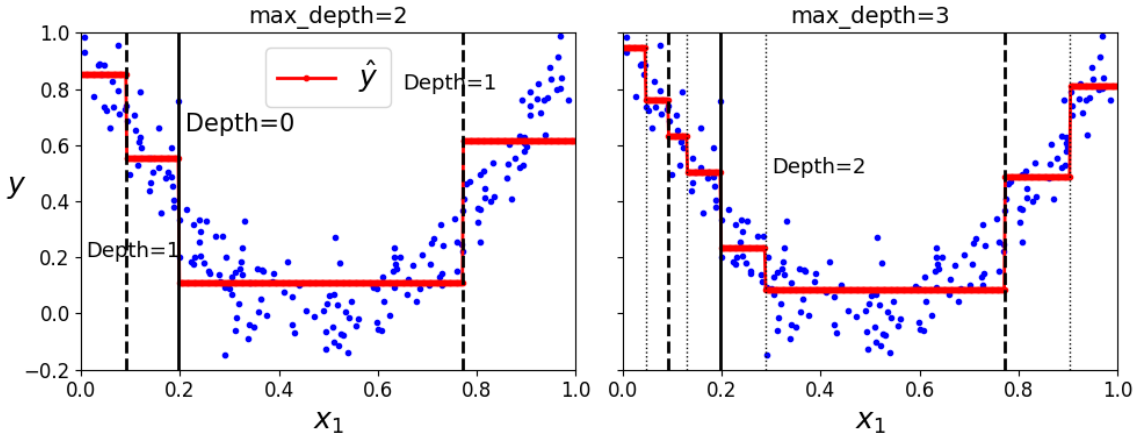
그림 6-4 회귀 결정 트리

→ 앞서 만든 분류 트리와 매우 비슷해 보이지만, 주요한 차이는 각 노드에서 클래스를 예측하는 대신 어떤 값을 예측한다는 점이다.

ex) $x_1 = 0.6$인 샘플의 클래스를 예측한다고 가정해보자.

루트 노드부터 시작해서 트리를 순회하면 결국 $value = 0.111$ 인 리프 노드에 도달한다. 이 리프 노드에 있는 110개 훈련 샘플의 평균 타깃값이 예측값이 된다. 이 예측값을 사용해 110개 샘플에 대한 평균제곱오차(MSE)를 계산하면 0.015가 된다.

이 모델의 예측은 다음과 같다.



→ `max depth=3`으로 설정하면 오른쪽 그래프와 같은 예측을 얻게 된다. 각 영역의 예측값은 항상그 영역에 있는 타깃값의 평균이다. 알고리즘은 예측값과 가능한 많은 샘플이 가까이 있도록 영역을 분할한다.

- **비용함수**

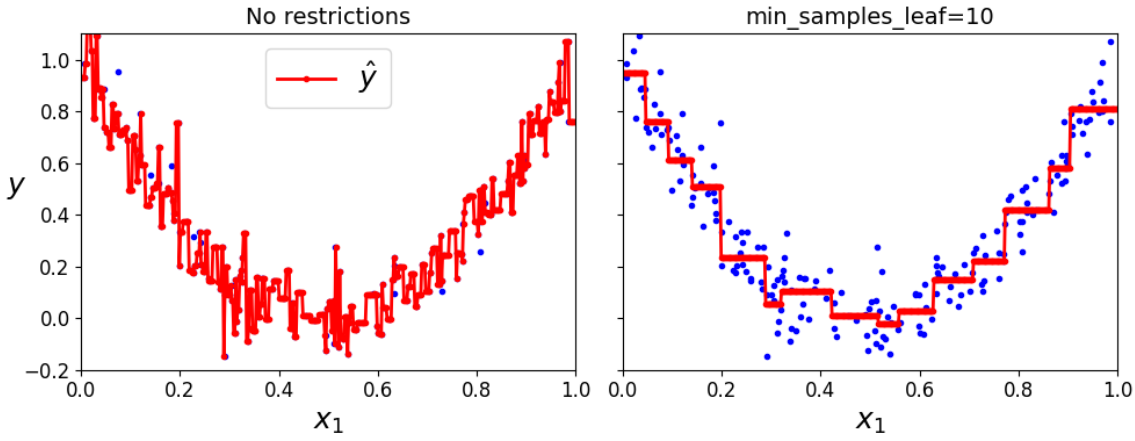
  : CART 알고리즘은 훈련 세트를 불순도를 최소화하는 방향으로 분할하는 대신 평균제곱오차(MSE)를 최소화하도록 분할하는 것을 제외하고는 앞서 설명한 것과 거의 비슷하게 작동한다. 알고리즘이 최소화하기 위한 비용 함수는 다음과 같다.

식 6-4 회귀를 위한 CART 비용 함수

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

여기서

$$\begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)} \right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$



회귀 작업 에서도 결정 트리가 과대 적합되기 쉽다. 규제가 없다면 (즉, 기본 매개변수를 사용) [그림 6-6]의 왼쪽과 같은 예측을 하게 된다. 이 예측은 확실히 훈련 세트에 과대 적합 되었다.

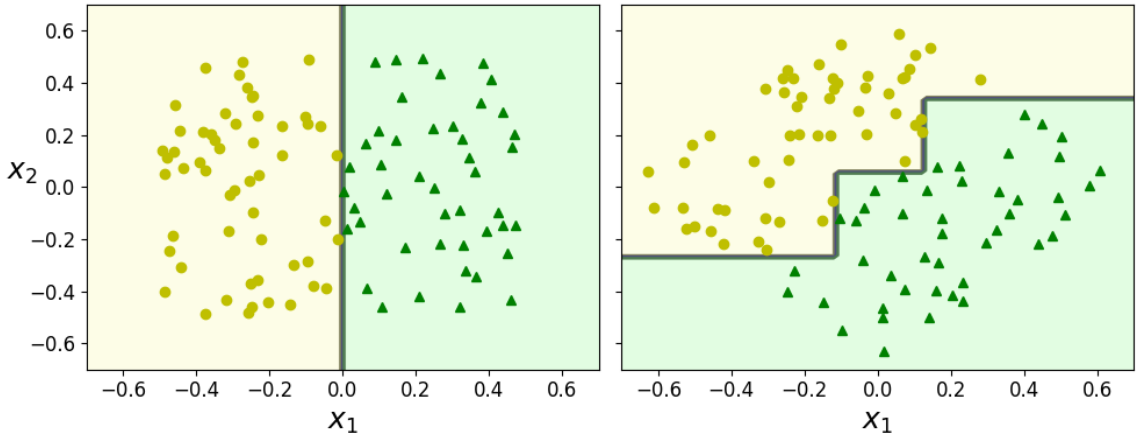
`min_samples_leaf=10`으로 지정하면, [그림 6-6] 의 오른쪽 그래프처럼 훨씬 그럴싸한 모델을 만들어준다.

▼ 6.9 불안전성

(1) 회전에 민감

결정 트리는 이해하고 해석하기 쉽고, 사용하기 편하고, 여러 용도로 사용할 수 있으며, 성능도 뛰어나다. 하지만 결정 트리는 계단 모양의 결정 경계를 만든다.(모든분할은축에 수직입) . 그래서 훈련 세트의 회전에 민감하다.

ex) 간단한 선형으로 구분될 수 있는 데이터셋



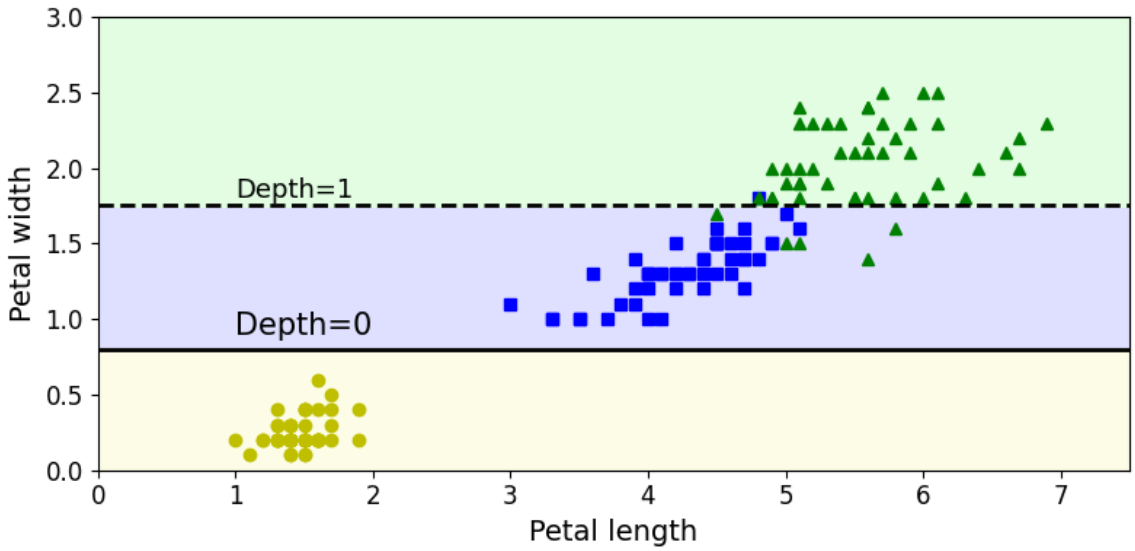
→ 왼쪽의 결정 트리는 쉽게 데이터셋을 구분하지만, 데이터셋을 45’ 회전한 오른쪽의 결정 트리는 불필요하게 구불구불하다. 두 결
정 트리 모두 훈련 세트를 완벽하게 학습하지만 오른쪽 모델은 잘 일반화되지 않는다.

이런 문제를 해결하는 한 가지 방법은 훈련 데이터를 더 좋은 방향으로 회전시키는 PCA 기법이 있다.

(2) 작은 변화에 민감

결정 트리의 주된 문제는 훈련 데이터에 있는 작은 변화에도 매우 민감하다는 것이다.

ex) 훈련 세트에서 가장 넓은 Iris-Versicolor(꽃잎 길이가 4.8cm이고 너비가 1.8cm 인 것)를 제거하고 결정 트리를 훈련시키면 [그림 6-8J 과 같은 모델을 얻게 된다.



→ 이전에 만든 결정 트리 (그림 6-2) 와는 매우 다른 모습이다. 사실 사이킷런에서 사용하는 훈련 알고리즘은 확률적이기 때문에 (`random_state` 매개변수를 지정하지 않으면) 같은 훈련 데이터에서도 다른 모델을 얻게 될 수 있다.

-

(회전 같은) 데이터셋의 작은 변화가 매우 다른 결정 트리를 만들었다. 사이킷런에서 사용하는 CART 훈련 알고리즘은 확률적이기 때문에 동일한 데이터에서 같은 모델을 훈련하여 매번 매우 다른 모델을 만들 수 있다. 이를 확인하기 위해 random_state를 다른 값으로 지정해 보자.

```
tree_clf_tweaked = DecisionTreeClassifier(max_depth=2, random_state=40)
tree_clf_tweaked.fit(X, y)

#<그림 6-8. 훈련 세트의 세부사항에 민감한 결정 트리> 생성 코드
plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_tweaked, X, y, legend=False)
plt.plot([0, 7.5], [0.8, 0.8], "k-", linewidth=2)
plt.plot([0, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.text(1.0, 0.9, "Depth=0", fontsize=15)
plt.text(1.0, 1.80, "Depth=1", fontsize=13)

save_fig("decision_tree_instability_plot")
plt.show()
```

```
#회전
angle = np.pi / 180 * 20
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
Xr = X.dot(rotation_matrix)

tree_clf_r = DecisionTreeClassifier(random_state=42)
tree_clf_r.fit(Xr, y)

plt.figure(figsize=(8, 3))
plot_decision_boundary(tree_clf_r, Xr, y, axes=[0.5, 7.5, -1.0, 1], iris=False)
```

```
plt.show()
```