

6장 결정트리

주요내용

- 결정트리 훈련과 활용
- CART 알고리즘
- 지니 불순도 vs. 엔트로피
- 결정트리 규제
- 회귀 결정트리
- 결정트리 단점

6.1 결정트리 훈련과 활용

결정트리 훈련

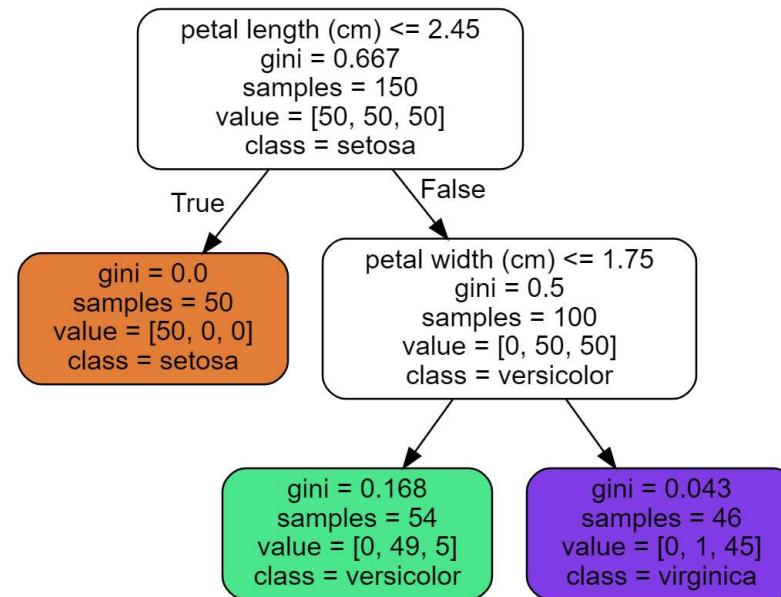
- 결정트리 방식의 최대 장점: 데이터 전처리 일반적으로 불필요. 필요한 경우도 존재.
- 사이킷런의 `DecisionTreeClassifier` 모델 활용
- 붓꽃 데이터 활용. 꽃잎의 길이와 너비 기준으로 분류.
- `max_depth=2`: 결정트리의 최대 깊이 지정. 여기서는 최대 2번의 데이터셋 분할 허용. 기본값은 `None`이며 무제한 데이터셋 분할 허용.

```
iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)", "petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

결정트리 시각화

- 사이킷런의 `export_graphviz()` 함수 활용
- pdf, png 등 많은 종류의 파일로 변환 가능



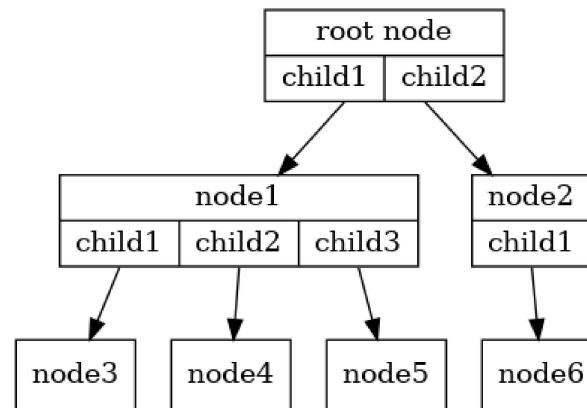
결정트리 노드의 속성

- `gini`: 노드의 지니 불순도
- `samples`: 노드에 속하는 샘플수
- `value`: 각각의 범주(클래스)에 속하면서 노드에 포함된 샘플 수. 타깃 정보 활용.
- `class`: 가장 높은 비율을 차지하는 범주. 비율이 동일한 경우 낮은 인덱스의 범주 선택.

트리

트리 rooted tree는 아래 성질을 만족하는 노드 node와 이음선 edge들로 구성됨.

- 루트로 사용되는 하나의 노드가 있음.
- 루트를 제외한 모든 노드는 부모 노드로부터 오는 진입 이음선으로 연결됨.
- 루트로부터 임의의 노트로 연결되는 경로가 유일하게 존재함.



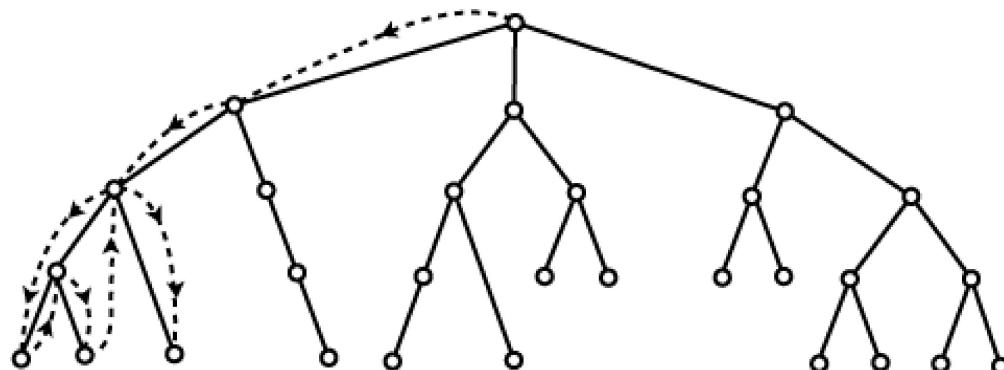
트리 구성 요소

- 루트_{root} 노드: 맨 상단에 위치한 노드. 부모 노드를 갖지 않음.
- 잎_{leaf} 노드: 더 이상의 가지분할이 발생하지 않는 노드. 자식 노드를 갖지 않음.

노드의 인덱스

노드의 인덱스는 깊이 우선 탐색(depth-first-search, DFS) 방식으로 이루어짐.

- 루트 노드에서 시작해서 갈 수 있는 한 가장 왼쪽 자식 노드로 탐색 진행
- 잎 노드에 다달한 경우 탐색 대상이 될 수 있는 자식 노드를 갖는 가장 가까운 조상 노드로 이동해서 깊이 우선 탐색 진행. 단, 한 번 탐색한 자식 노드는 무시.



지니 불순도

- G_i : i -번째 노드의 지니 불순도

$$G_i = 1 - \sum_{k=0}^{K-1} (p_{i,k})^2$$

- $p_{i,k}$ 는 i 번째 노드에 있는 훈련 샘플 중 범주 k 에 속한 샘플의 비율. K 는 범주의 개수.
- 예제: 깊이 2의 왼편 노드 G_3 의 지니 불순도

$$G_4 = 1 - (0/54)^2 - (49/54)^2 - (5/54)^2 = 0.168$$

- 예제: 깊이 1의 왼편 노드 G_1 의 지니 불순도. 세토사 품종으로만 구성되었음.

$$G_1 = 1 - (55/55)^2 - (0/55)^2 - (0/55)^2 = 0$$

범주 예측

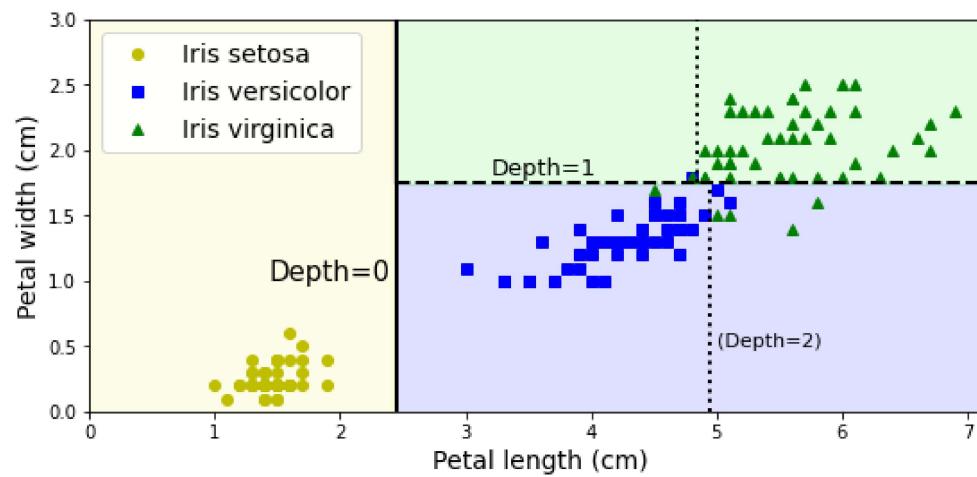
예제

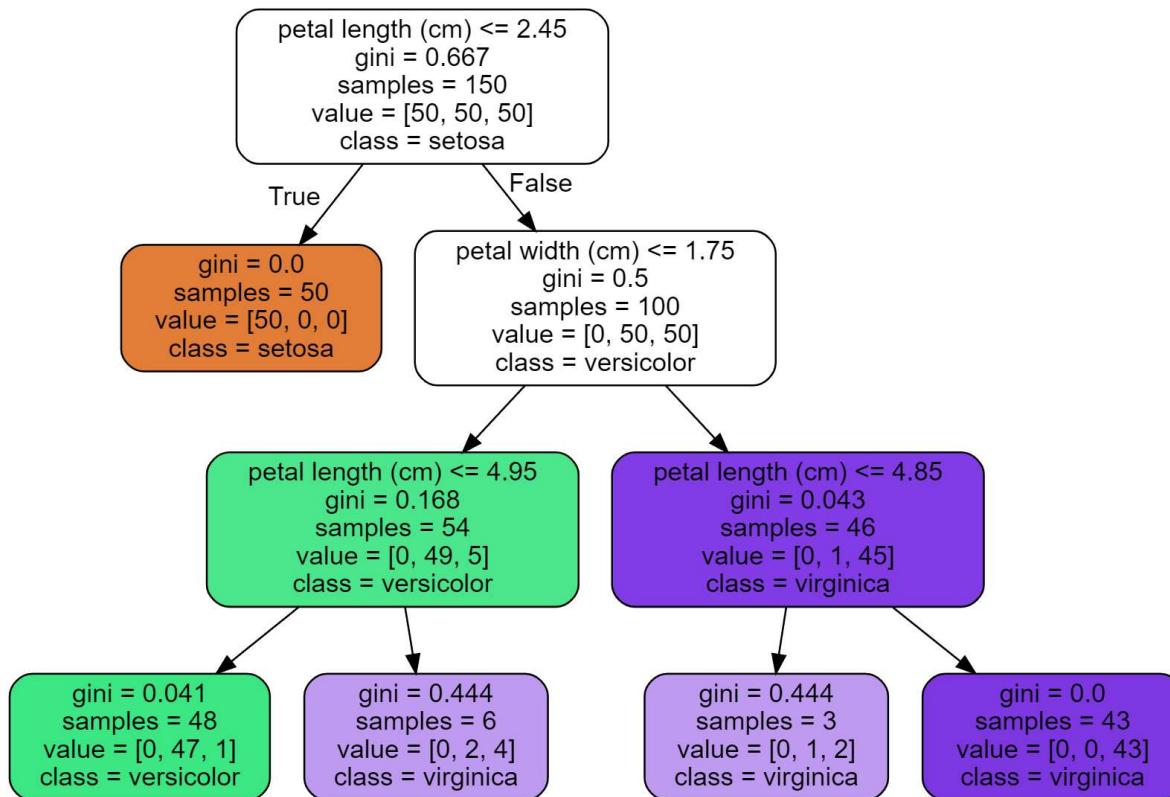
- 꽃잎 길이와 너비: 각각 5cm, 1.5cm
- 데이터가 주어지면 루트에서 시작
- 분할 1단계: 꽃잎 길이가 2.45cm 이하가 아니기에 오른편으로 이동.
- 분할 2단계: 꽃잎 너비가 1.75cm 이하이기에 왼편으로 이동. 버시컬러로 판정.

결정경계

아래 그림은 `max_depth=3` 으로 지정해서 학습한 결정트리의 결정경계를 보여줌.

- 1차 분할 기준: 꽃잎 길이 2.45cm. 트리 깊이 0에서 진행됨.
- 2차 분할 기준: 꽃잎 너비 1.75cm. 트리 깊이 1에서 진행됨.
- 3차 분할 기준: 꽃잎 길이 4.95cm. 트리 깊이 2에서 진행됨.
- 4차 분할 기준: 꽃잎 길이 4.85cm. 트리 깊이 2에서 진행됨.





범주에 속할 확률

- 주어진 샘플에 대해 예측된 노드에 속한 샘플들의 범주별 비율
- `max_depth=2`로 훈련된 `tree_clf` 모델은 예를 들어 꽃잎 길이와 너비가 각각 5cm, 1.5cm인 붓꽃을 깊이 2의 왼편 잎 노드인 G_3 에 포함시킴.
- G_3 노드에 포함된 샘플들의 범주별 비율: (세토사, 버시컬러, 버지니카)
$$(0/54, 49/54, 5/54) = (0, 0.907, 0.093)$$
- 버시컬러에 속할 확률이 90.7%로 가장 높기에 해당 샘플은 버시컬러로 예측됨.

`predict_proba()` vs. `predict()`

`predict_proba()` 메서드: 지정된 샘플의 범주별 추정 확률을 계산

```
>>> tree_clf.predict_proba([[5, 1.5]]).round(3)
array([[0...., 0.907, 0.093]])
```

`predict()` 메서드: 품종 범주를 예측하며, 가장 높은 추정 확률을 갖는 품종으로 지정.

```
>>> tree_clf.predict_proba([[5, 1.5]]).round(3)
array([1])
```

`max_depth=3` 인 경우 다르게 예측됨

```
>>> tree_clf_deeper.predict_proba([[5, 1.5]]).round(3)
array([[0...., 0.333, 0.667]])
```

```
>>> tree_clf_deeper.predict([[5, 1.5]]).round(3)
array([2])
```

6.2 CART 훈련 알고리즘

CART(Classification and Regression Tree) 분류 알고리즘의 비용 함수

- 각 노드에서 아래 비용함수를 최소화 하는 특성 k 와 해당 특성의 임곗값 t_k 를 결정 해서 사용함.
 - $m, m_{\text{left}}, m_{\text{right}}$: 각각 부모와 양쪽 자식 노드에 속한 샘플 수
 - $G_{\text{left}}, G_{\text{right}}$: 두 자식 노드의 지니 불순도

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

- $J(k, t_k)$ 가 작을 수록 불순도가 낮은 두 개의 부분집합으로 분할됨
- **참고:** 탐욕적 알고리즘 사용. 해당 노드를 기준으로 지니 불순도가 가장 낮은, 즉, 가장 순수한(pure) 두 개의 부분집합으로 분할함. 최적의 분할이란 보장은 없지만 일반적으로 적절한 성능을 보임.
- 분할 과정 반복: `max_depth` 등 규제의 한계에 다다르거나 더 이상 불순도를 줄이는 분할이 불가능할 때까지 진행.

CART 알고리즘의 계산 복잡도

- 최적의 결정트리를 찾는 문제는 NP-완전(NP-complete)임.
- 이런 문제의 시간 복잡도는 $O(\exp(m))$
- 매우 작은 훈련셋에 대해서도 제대로 적용하기 어려움

결정트리 모델의 예측 시간 복잡도

- 학습된 결정트리가 예측에 필요한 시간: $O(\log m)$
- 훈련 샘플 수 m 에만 의존하며 매우 빠름. 각 노드에서 하나의 특성만 분류기준으로 사용되기에 특성 수와 무관하기 때문임.

CART 알고리즘의 시간 복잡도

- 훈련 샘플이 크기순으로 정렬된 경우 (n, m 은 각각 특성 개수와 샘플 개수를 나타냄):
 - 각 노드에서 분류하는 데 걸리는 시간: $O(n \cdot m \cdot \log(m))$
 - 결정트리를 완성하는 데 걸리는 시간: $O(n \cdot m^2 \cdot \log(m))$
 - 규제가 있는 경우 좀 더 빨라짐.
- 훈련셋의 크기가 몇 천보다 크면 정렬 자체가 오래 걸림. 가장 빠른 정렬 알고리즘의 복잡도가 $O(m \log m)$ 정도임.

지니 불순도 vs. 엔트로피

- `DecisionTreeClassifier` 의 `criterion="entropy"` 옵션 설정:
 - gini 불순도 대신에 샘플들의 무질서 정도를 측정하는 엔트로피 사용
 - 지니 불순도를 사용할 때와 비교해서 큰 차이가 나지 않음.
 - 엔트로피 방식이 노드를 보다 균형 잡힌 두 개의 자식 노드로 분할함.
- i -번 인덱스 노드의 엔트로피

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

- 두 방식의 큰 차이가 없고 지니 불순도 방식이 보다 빠르게 훈련되어 기본값으로 지정됨.

규제 하이퍼파라미터

비파라미터 모델

- 결정트리 모델은 데이터에 대한 어떤 가정도 하지 않음.
- 예를 들어, 노드를 분할할 수 있는 자유도 degree of freedom에 대한 제한이 기본적으로 없음.
- 이런 모델을 **비파라미터 모델** nonparametric model이라 함.
- 규제를 가하지 않으면 과대적합 위험 매우 높음

사이킷런 `DecisionTreeClassifier` 규제 하이퍼파라미터

- 규제 강화 방법
 - `min_` 접두사 사용 규제: 값을 키울 것
 - `max_` 접두사 사용 규제: 값을 감소시킬 것

하이퍼파라미터	기능
<code>max_depth</code>	결정트리의 높이 제한
<code>min_samples_split</code>	노드 분할해 필요한 최소 샘플 개수
<code>min_samples_leaf</code>	잎 노드에 포함된 최소 샘플 개수
<code>max_leaf_nodes</code>	최대 잎 노드 개수
<code>max_features</code>	분할에 사용되는 특성 개수

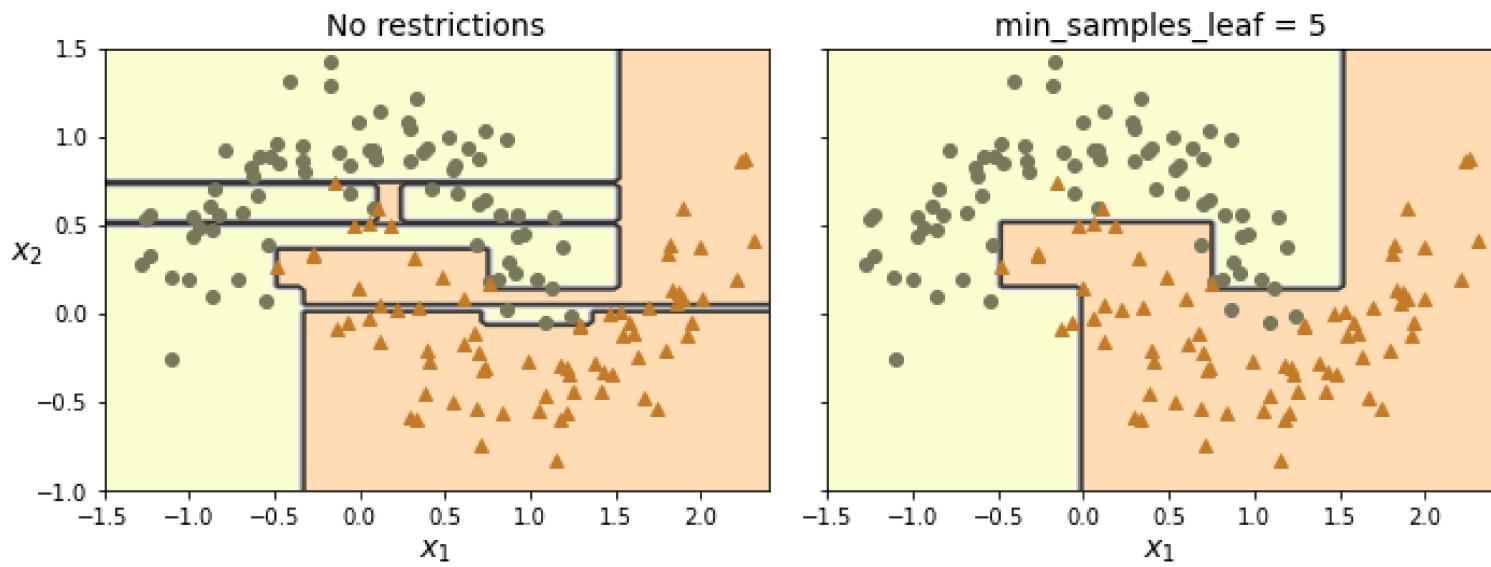
예제: 규제 적용

- 예제: 초승달 데이터셋에 대한 결정트리 모델 학습
 - 왼편: 규제 전혀 없음. 보다 정교하며 과대적합됨.
 - 오른편: `min_samples_leaf=5`. 일반화 성능이 보다 좋음.

```
from sklearn.datasets import make_moons

X_moons, y_moons = make_moons(n_samples=150, noise=0.2, random_state=42)

tree_clf1 = DecisionTreeClassifier(random_state=42)
tree_clf2 = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
tree_clf1.fit(X_moons, y_moons)
tree_clf2.fit(X_moons, y_moons)
```



6.3 회귀 결정트리

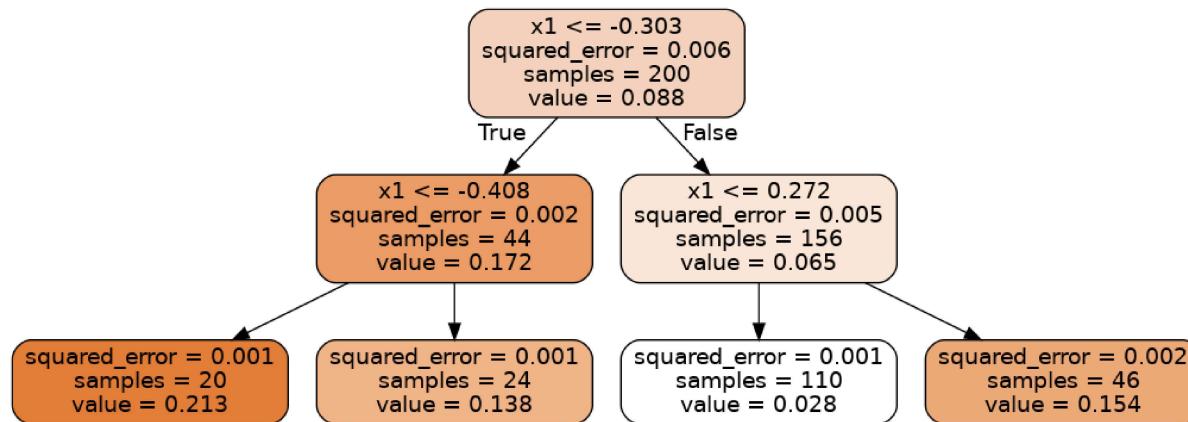
사이킷런의 DecisionTreeRegressor 예측기 활용

- 결정트리 알고리즘 아이디어를 거의 그대로 이용하여 회귀 문제에 적용 가능

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

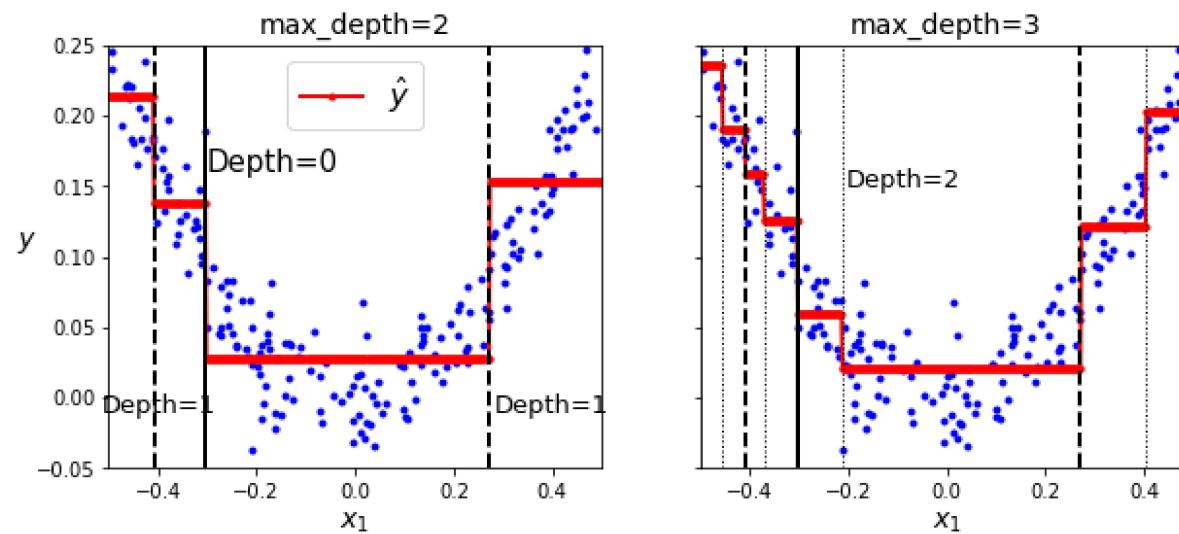
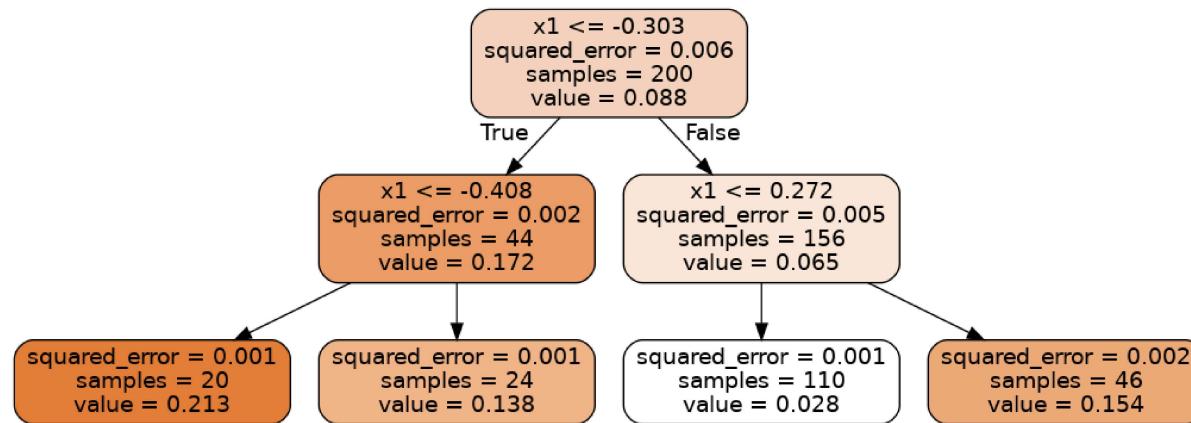
예제: 잡음이 포함된 2차 함수 형태의 데이터셋

- 왼편 그림 설명



- 각 노드에 포함된 속성
 - samples : 해당 노드에 속한 훈련 샘플 수
 - value : 해당 노드에 속한 훈련 샘플의 평균 타깃값
 - squared_error : 해당 노드에 속한 훈련 샘플의 평균제곱오차(MSE)
 - 오차 기준은 value 사용.

- 왼편 그림 설명



회귀용 CART 알고리즘과 비용함수

- 분류의 경우처럼 탐욕적으로 아래 비용함수를 최소화 하는 특성 k 와 해당 특성의 임계값 t_k 을 결정함:
 - MSE_{node} : 해당 노드의 평균제곱오차(MSE).
 - m_{node} : 해당 노드에 속하는 샘플 수
 - $y^{(i)}$: 샘플 i 에 대한 실제 타깃값

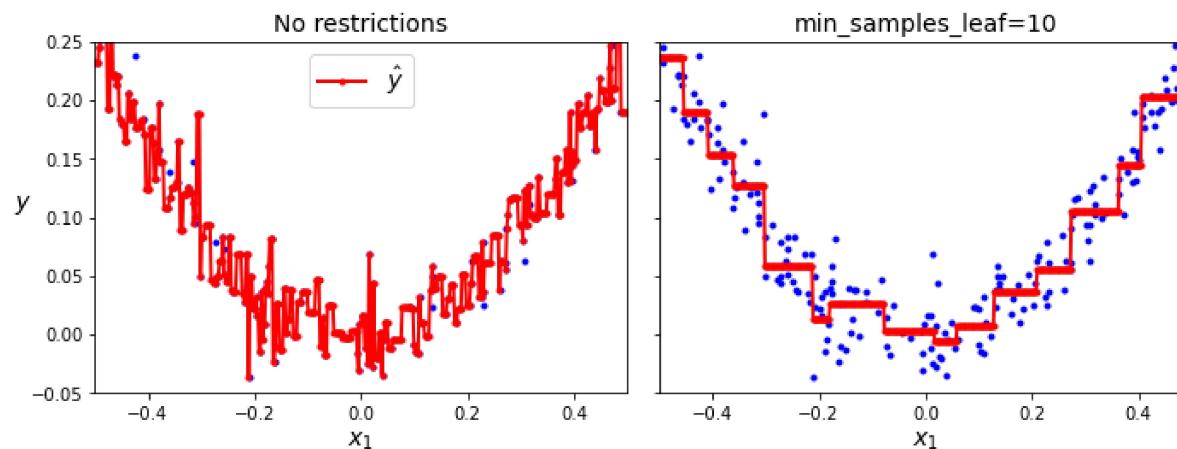
$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}}$$

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

$$\hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)}$$

규제

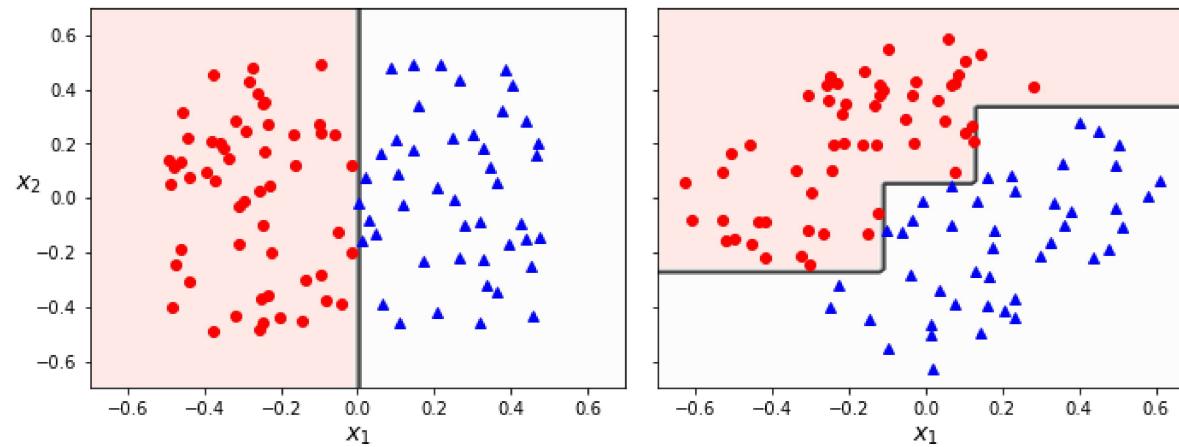
- 분류의 경우처럼 규제가 없으면 과대적합 발생할 수 있음.
- 왼편: 규제가 없는 경우. 과대적합 발생
- 오른편: `min_samples_leaf=10`



6.4 결정트리 단점

단점 1: 훈련셋 회전 민감도

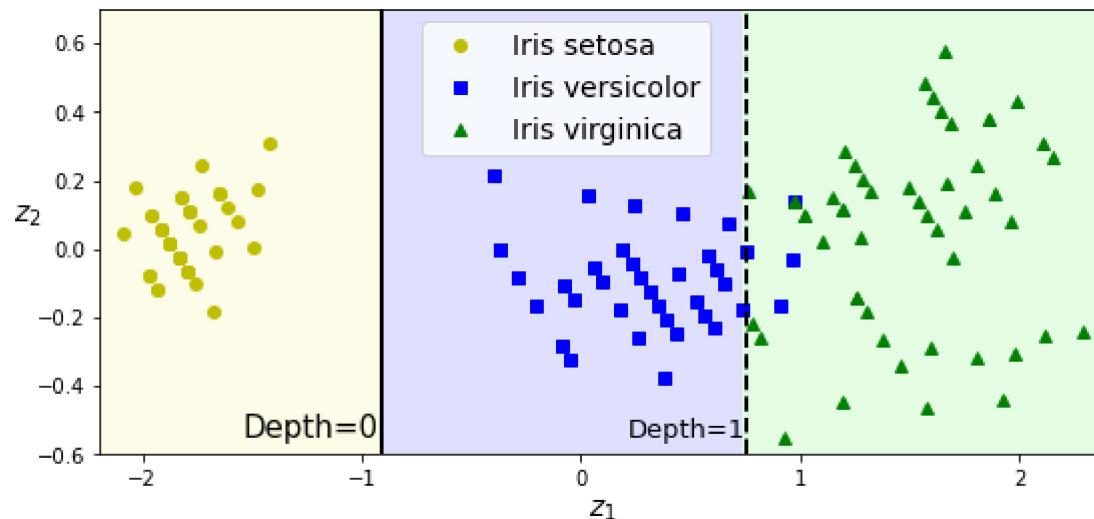
- 결정트리 알고리즘은 성능이 매우 우수하지만 기본적으로 주어진 훈련셋에 민감하게 반응함.
- 결정트리는 항상 축에 수직인 분할을 사용. 따라서 조금만 회전을 가해도 결정 경계가 많이 달라짐
- 예제: 오른편 그래프: 왼편 그래프를 45도 회전시킨 훈련셋 학습



예제: PCA() 모델 기법 적용 데이터 변환

- PCA 기법: 데이터셋을 회전시켜서 특성들 사이의 연관성을 약화시킴
- 예제: PCA 기법으로 회전시킨 붓꽃 데이터셋에 분류 결정트리를 훈련시킨 결과

```
pca_pipeline = make_pipeline(StandardScaler(), PCA())
X_iris_rotated = pca_pipeline.fit_transform(X_iris)
tree_clf_pca = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_pca.fit(X_iris_rotated, y_iris)
```



단점 2: 높은 분산

- 훈련 데이터의 작은 변화에도 매우 민감함.
- `random_state`를 지정하지 않으면서 동일한 모델을 훈련 시키면 다른 결과 나옴(아래 그래프 참고)
- 많은 트리에서 만든 예측값의 평균을 활용 추천(7장 램덤포레스트 모델 참고)

