CH6. 결정트리

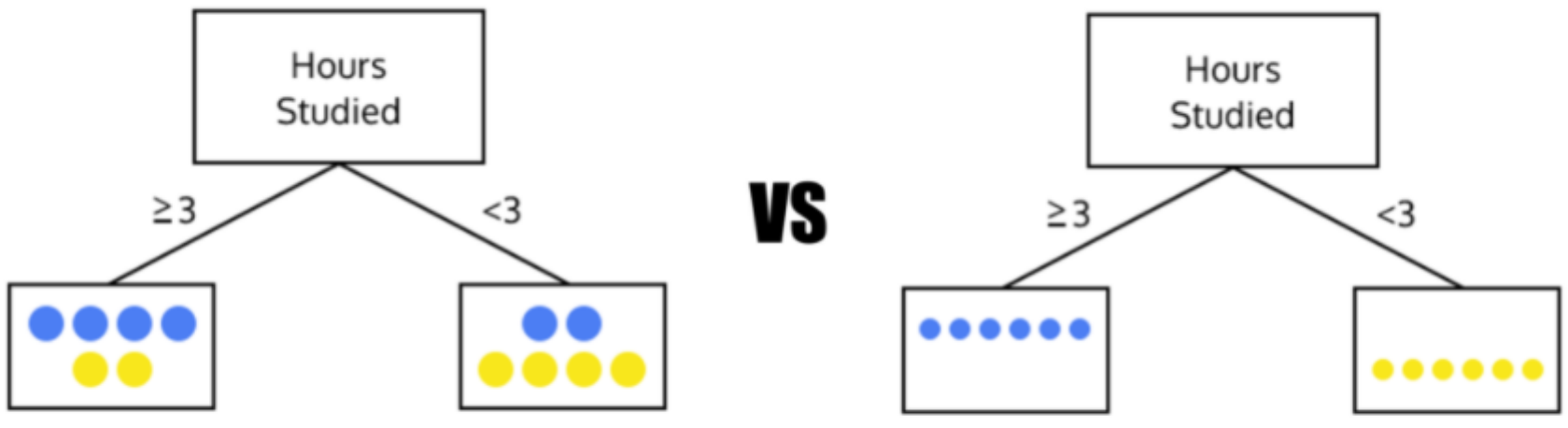
http://hleecaster.com/ml-decision-tree-concept/

📖 6.1 결정트리 학습과 시각화

⚬ 결정트리는 분류과 회귀작업 그리고 다중출력 작업도 가능한 다재다능한 머신러닝 알고리즘 + 매우 복잡한 데이터셋도 학습할 수 있는 강력한 알고리즘 + 랜덤 포레스트(최근 가장 강력한 머신러닝 알고리즘)의 기본 구성요소

⚬ 일종의 스무고개 같은 알고리즘이라 생각하면 됨!

- **지니 불순도**



일단 왼쪽의 그림은 데이터 분할이 깔끔하지 않다. 우리의 목표는 오른쪽 그림처럼 불순물(?) 없이 완전 깔끔하게 골라내는 거다.

여기서 불순물이라는 표현을 썼는데, 이 불순물이 어느정도 포함되어 있는지 확인할 수 있는 지표로 **지니 불순도(Gini Impurity)** 값을 확인해볼 수 있다.

→ 계산법 : 1에서 '전체 데이터 개수 중 각 레이블이 차지하는 개수의 비율'을 제곱해서 빼주기

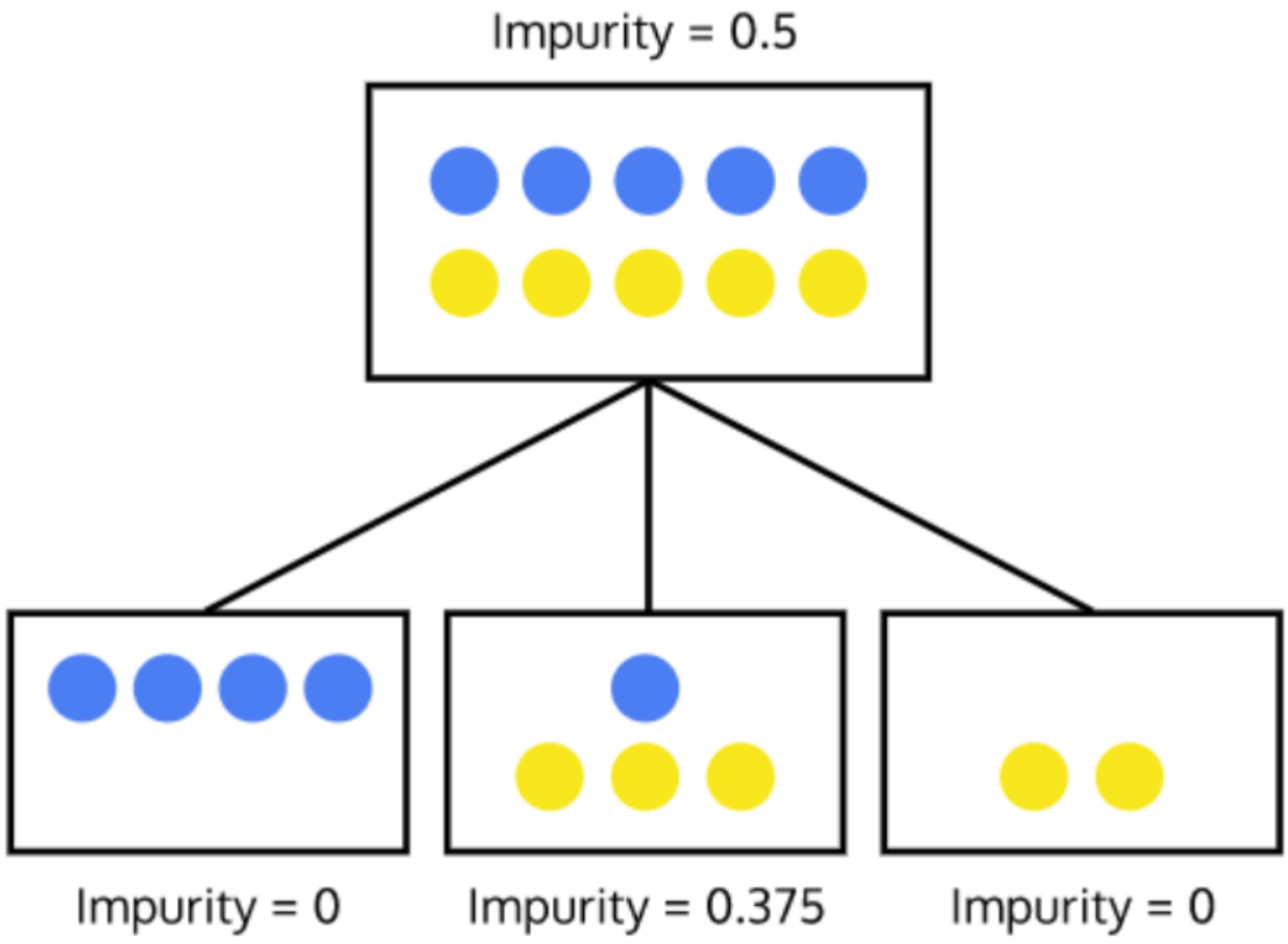
예를 들어 총 4개 데이터 세트에서 레이블 A가 3개, B가 1개 포함되어 있다면 이렇게 계산하면 된다.

$$1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.375$$

만약 단 하나의 레이블로 퓨어하게 구성되어 있으면 지니 불순도 값은 0이 된다. 결국 좋은 질문을 거쳐 분할된 데이터 세트는 지니 불순도 값이 작다는 것을 알 수 있다.

- **정보획득량**

지니 불순도를 구할 수 있다면, 이를 통해 어떤 질문에서 얻을 수 있는 정보 획득량(Information Gain)을 계산할 수 있다.



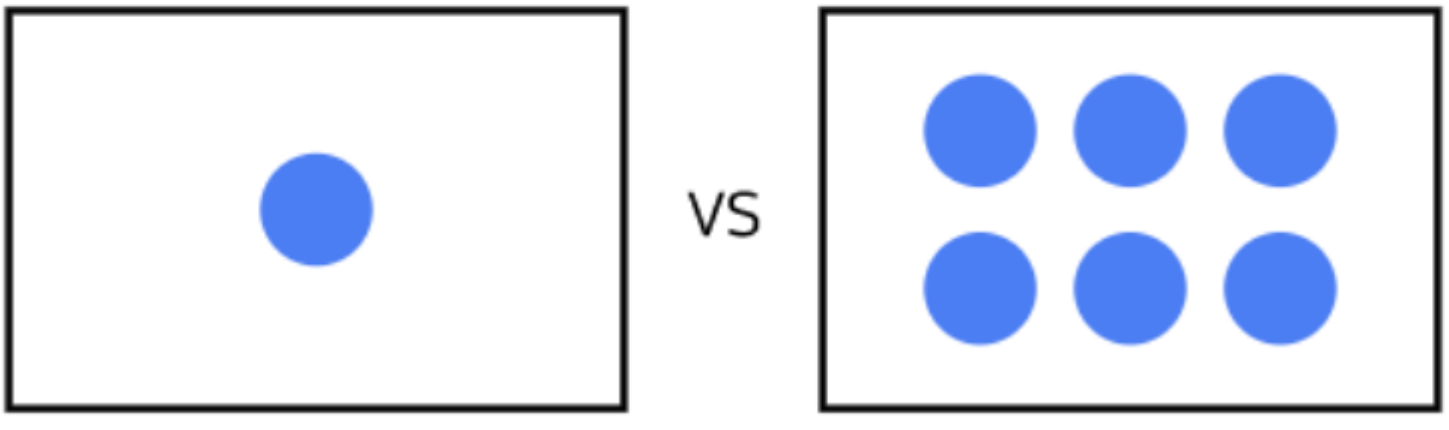
위처럼 불순도 0.5의 데이터 세트를 불순도 각각 0, 0.375, 0을 가진 3개의 데이터 세트로 분할했을 때, 여기서 얻은 정보 획득량은 다음과 같이 계산할 수 있다.

Information Gain = 0.5 – (0 + 0.375 + 0) = 0.125

그냥 이전 단계 불순도에서 다음 단계의 불순도 합을 빼주는 거다. 즉, **분할된 데이터 세트들의 불순도가 작을수록 정보 획득량은 증가**한다. 그러나 이렇게 하위 데이터 세트들의 불순도만으로는 정보 획득량을 설명하기 부족한 측면이 있다. 그래서 **가중치**(weight)를 적용하곤 한다.

Weighted Information Gain

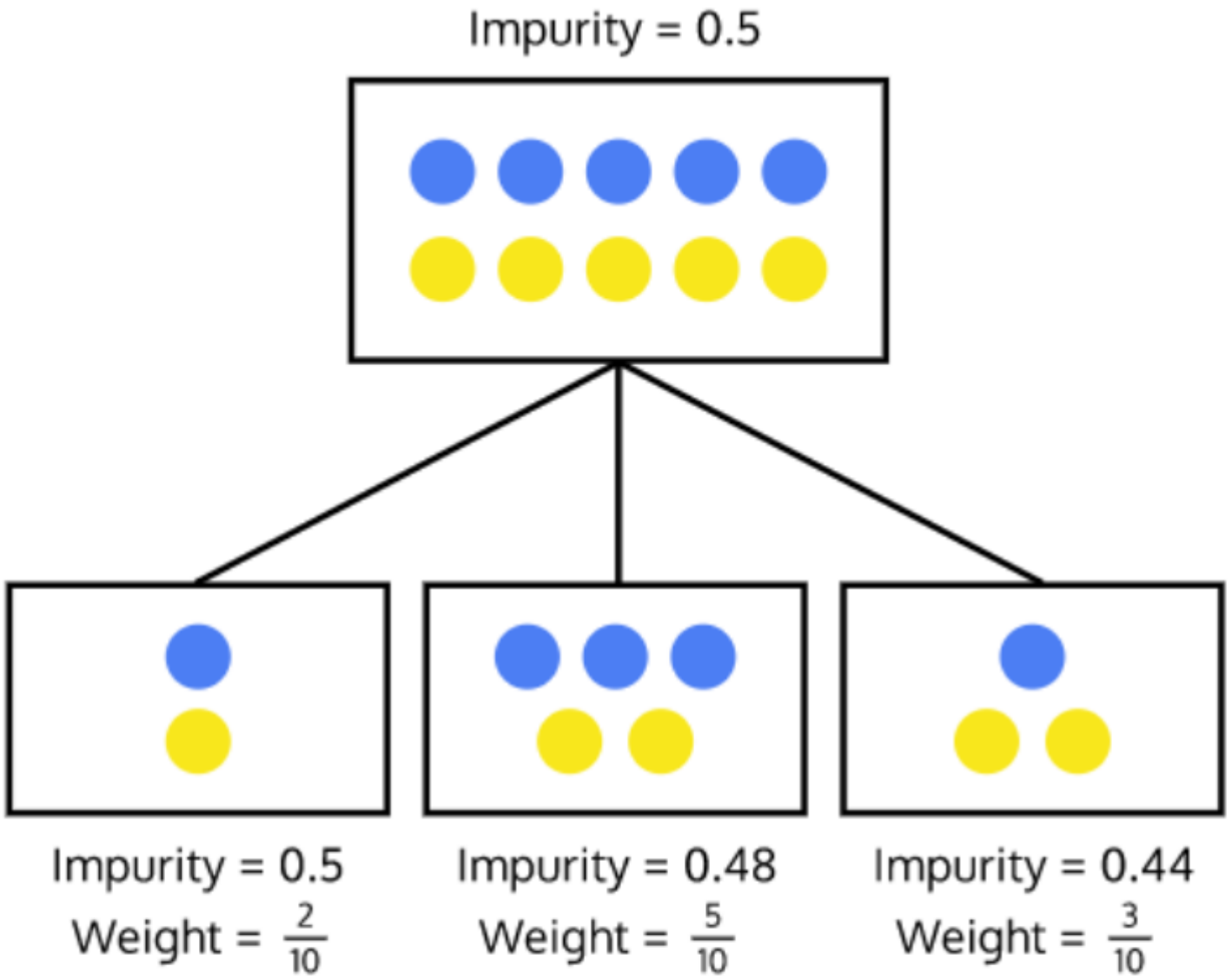
아래 그림을 보자.



두 개의 데이터 세트 모두 불순도는 0이지만, 오른쪽 데이터 세트가 더 의미있는 것처럼 보인다. 그 이유가 뭘까. **데이터 개수가 충분히 많고, 따라서 이 분류가 우연이 아니라고 확신할 수 있기 때문**이다.

아무튼 이제 단순한 불순도뿐만 아니라 데이터 세트의 크기도 중요하다는 걸 알았으니 **생성된 데이터 세트의 크기에 따라 가중치가 적용된 정보 획득량 (Weighted Information Gain)을 계산**해볼 수 있을 거다.

계산법



이렇게 **분할하기 전 데이터에 비해 분할 후 생성된 데이터의 크기(비율)에 따라 가중치를 구해놓고 이를 불순도에 곱해서 정보 획득량을 구하면 된다.**

Information Gain = 0.5 – ((2/10)*0.5 + (5/10)*0.48 + (3/10)*0.44) = 0.026

→ 데이터 세트의 크기가 작을 수록 그 불순도의 영향력도 작아진다.

- **재귀적 트리빌딩**

의사결정 트리에서는 **정보 획득량(Information Gain)**이 **큰 순서대로 질문을 배치**하는 게 중요하다. (상식적인 거다. 스무고개를 하더라도 크리티컬한 질문을 먼저 하는 게 좋지 않은가.) 그리고 그 속성에 대해 어느 기준으로 나누는 게 좋을지 **반복적으로 적용**해보면서 최적의 트리를 찾게 된다. 이를 재귀(recursive) 알고리즘이라고 한다. 아무튼 이렇게 반복하면서 찾으면서 더 이상 정보 획득량이 없을 때 재귀(recursion)를 중단하게 된다. 더 이상은 순수한 하위 집합을 만드는 방법을 찾을 수 없을 때까지 하는 거다.

- **사이킷런 사용법**

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(training_points, training_labels)
```

학습 데이터를 기반으로 트리를 생성할 때는 `.fit()` 메서드를 사용하면 끝이다. 다만, 의사결정 트리에서는 만약 데이터가 문자열로 이루어져 있다면 이걸 숫자로 map해서 넣어주는 게 좋다. 예를 들어 {“low”: 1, “mid”: 2, “high”: 3} 이런 식으로.

새로운 데이터가 있을 때 예측을 해볼 수도 있고,

```
predictions = classifier.predict(test_data)
```

미리 테스트 세트를 나눠 놓았다면 분류 정확도를 확인해볼 수도 있다.

```
print(classifier.score(test_data, test_labels))
```

- **의사결정 나무의 한계**

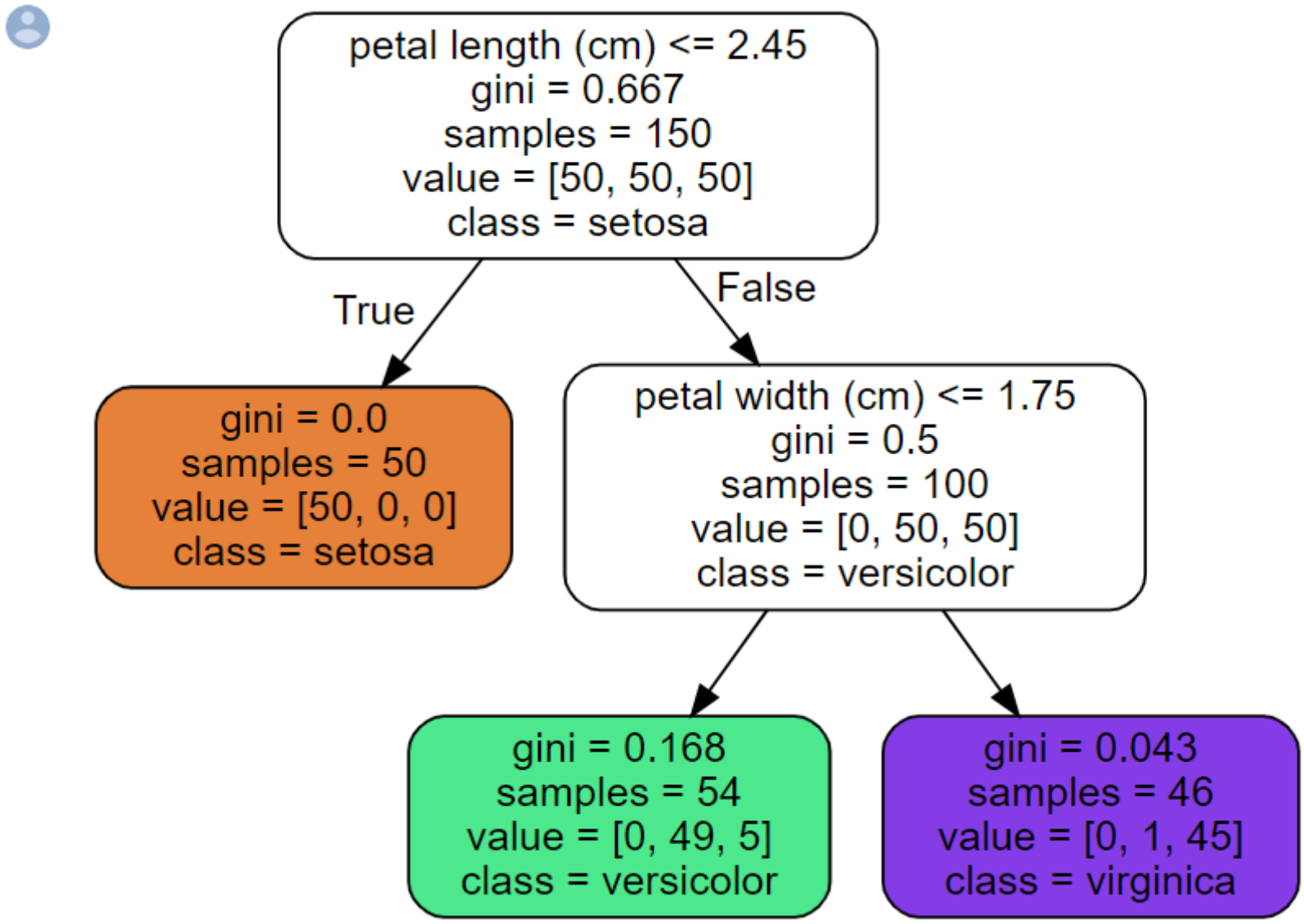
1) 트리를 빌딩, 생성하는 greedy 방식에서의 한계

: 의사결정 나무는 지금 상황에서 어떤 속성을 가지고 분할해야 정보 획득량이 가장 클 것인지 확인해서, 그 속성에 따라 데이터를 분할한다. 즉, 데이터를 먼저 이렇게 나누었을 때 나타날 파급 효과를 고려하지 않는 거다. 그 순간만 보면 최선의 분할이 아니지만 **이어서 나타나는 속성들에서 더 나은 분할을 발견하면 결과적으로 더 좋은 트리를 만들 수도 있는데, 순간의 선택 때문에 그 가능성을 배제**해버린다는 뜻이다.

2) 의사결정 나무가 학습 데이터에 오버피팅 한다

: 트리의 구조가 훈련 데이터에 너무 의존적이라 현실의 데이터를 정확하게 나타내지 못한다는 뜻이다. 일반적으로 나무가 커질수록 학습 데이터에 알맞게 모델이 만들어져서 현실 데이터로 일반화(generalization)하기 어려워지는 경향이 있다. 이걸 해결하려면 결국 나무를 잘라내서 크기를 줄여야 한다. 일명 가지치기(pruning).

- **붓꽃 데이터 결정트리 예시 (export_graphviz 함수로 출력가능)**



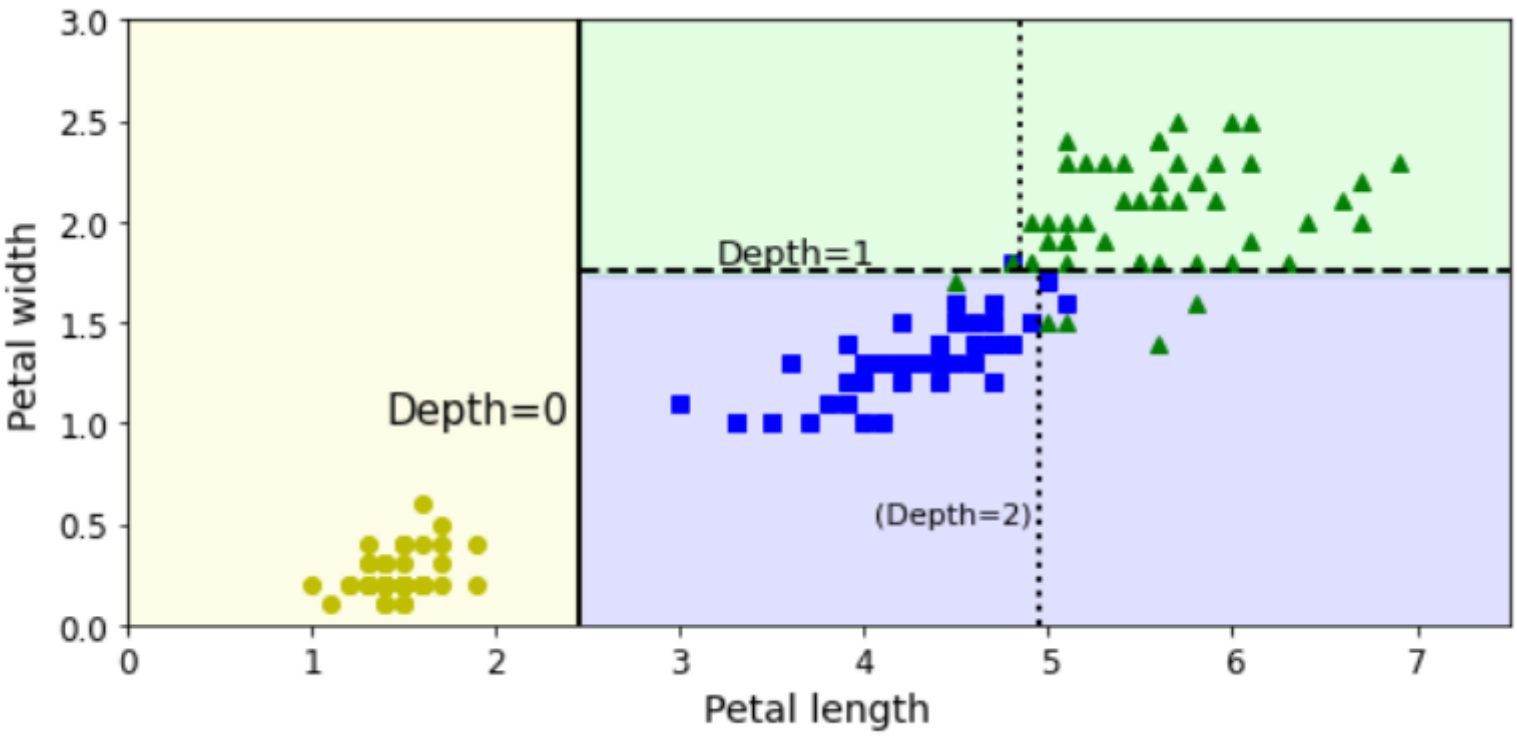
📖 6.2 예측하기

새로 발견한 붓꽃의 품종을 분류한다고 가정 (위의 결정트리 사진 참고)

- 루트노드 : 맨 꼭대기 노드
- 리프노드 : 자식을 가지지 않는 노드. 이 노드에 도달하면 추가적인 검사를 하지 않고, 노드에 있는 예측 클래스를 보고 결정트리가 새로 발견한 꽃의 품종을 결정지음

- 노드의 sample 속성 : 얼마나 많은 훈련 샘플이 적용되었는지 헤아린인 것이다.
- 노드의 value 속성 : 노드에서 각 클래스에 얼마나 많은 훈련 샘플이 있는지 알려준다.
- 노드의 gini 속성 : 불순도를 측정
- 불순도 : 한 노드의 모든 샘플이 같은 클래스에 속해있다면 이 노드를 순수(gini=0) 하다고 한다. 예를들어 깊이 1의 왼쪽 노드는 Iris-Setosa 훈련 샘플만 가지고 있으므로 순수노드이다.

- **결정 트리의 결정경계**



📖 6.3 클래스 확률 추정

결정트리는 한 샘플이 특정 클래스 k에 속할 확률을 추정할수도 있다. 샘플에 대해 리프노드를 찾기 위해 트리를 탐색하고 그 노드에 있는 클래스 k의 훈련샘플 비율을 반환한다. 예를들어 길이가 5cm이고 너비가 1.5cm인 꽃잎을 발견했다고 가정하면, 이에 해당하는 리프노드는 깊이 2에서 왼쪽 노드 이므로 결정트리는 그에 해당하는 확률을 출력한다. 즉 Setosa는 0% (**0**/54) , Versicolor는 90.7% (**49**/54) , Virginica는 (**5**/54) 이다. 만약 클래스를 하나 예측한다면 가장 높은 확률을 가진 Versicolor 를 출력하게 될 것이다.

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


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

📖 6.4 CART 훈련 알고리즘

사이킷런은 결정트리를 훈련시키기 위해, 즉 트리를 성장시키기 위해 CART 알고리즘을 사용한다. 먼저 훈련 세트를 하나의 특성 k의 임곗값 tk를 사용해 두개의 서브셋으로 나눈다. 이때 k와 tk는 크기에 따라 가중치가 적용된 가장 순수한 서브셋으로 나눌 수 있는 (k,tk) 짝을 찾는다. 이 알고리즘이 최소화 해야하는 비용함수는 다음과 같다.

식 6-2: 분류에 대한 CART 비용 함수

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

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

CART 알고리즘이 훈련세트를 성공적으로 둘로 나누었다면 같은 방식으로 서브셋을 또 나누고 그 다음엔 서브셋의 서브셋을 나누고 이런 식으로 계속 반복한다.

📖 6.5 계산 복잡도

예측하려면 결정트리를 루프노드에서부터 리프노드까지 탐색해야 한다. 일반적으로 결정 트리는 거의 균형을 이루고 있기 때문에 결정트리를 탐색하기 위해선 약 O(log2(m)) 개의 노드를 거쳐야 한다. 각 노드는 하나의 특성 값만 확인하기 때문에 예측에 필요한 전체 복잡도는 특성수와 무관하다. 따라서 큰 훈련 세트를 다룰 때도 예측속도가 매우 빠르다.

📖 6.6 지니 불순도 또는 엔트로피?

기본적으로 지니 불순도가 사용되지만, criterion 매개변수를 "entropy"로 지정하여 엔트로피 불순도를 사용할 수 있다. (분자가 안정화되고 질서정연하면 엔트로피가 0에 가까움) 여기서는 모든 메시지가 동일할 때 엔트로피가 0이 된다. 머신러닝에서는 불순도의 측정방법으로 자주 사용된다. 예를들어 어떤 세트가 한 클래스의 샘플만 담고 있으면 엔트로피가 0이 된다. 지니불순도와 엔트로피는 비슷한 트리를 만들어내서 결과적으로 큰 차이는 없지만, 지니불순도가 계산이 빠르긴 하다. 그러나 다른 트리가 만들어지는 경우 지니 불순도가 가장 빈도 높은 클래스를 한쪽 가지로 고립시키는 경우가 있는 반면 엔트로피는 조금 더 균형잡힌 트리를 만든다.

식 6-1: 지니 불순도

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

식 6-3: 엔트로피 불순도

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

📖 6.7 규제 매개변수

선형 모델은 데이터가 선형일거라는 가정 즉 제약사항이 도입되는데, 결정트리는 훈련데이터에 대한 이런 제약사항이 거의 없다. 제한을 두지 않으면 트리가 훈련 데이터에 과대적합이 쉽게 된다. 결정트리처럼 훈련되기 전에 파라미터 수가 결정되지 않는 모델을 비파라미터 모델이라고 부른다. 모델 구조가 데이터에 따라 맞춰지기 때문에 자유롭다. 반면 선형모델 같은 파라미터 모델은 미리 정의된 모델 파라미터 수를 가지기 때문에 자유도는 제한되고, 대신 과대적합의 위험이 줄어든다.

결정트리의 자유도를 어느정도 제한하여 과대적합을 피할 수 있는 방법으로 사이킷런의 `max_depth` 매개변수가 있다. 이 매개변수 값을 줄이면 모델을 규제하게 되고, 과대적합의 위험이 감소한다. 이외에도 다음과 같은 결정 트리의 형태를 제한하는 다른 매개변수가 있다.

- `min_samples_split` : 분할 되기 위해 노드가 가져야 하는 최소 샘플 수
- `min_samples_leaf` : 리프노드가 가지고 있어야 할 최소 샘플수
- `min_weight_fraction_leaf` : `min_samples_split`와 같지만 가중치가 부여된 전체 샘플 수에서의 비율
- `max_leaf_nodes` : 리프노드의 최대 수
- `max_features` : 각 노드에서 분할에 사용할 특성의 최대수

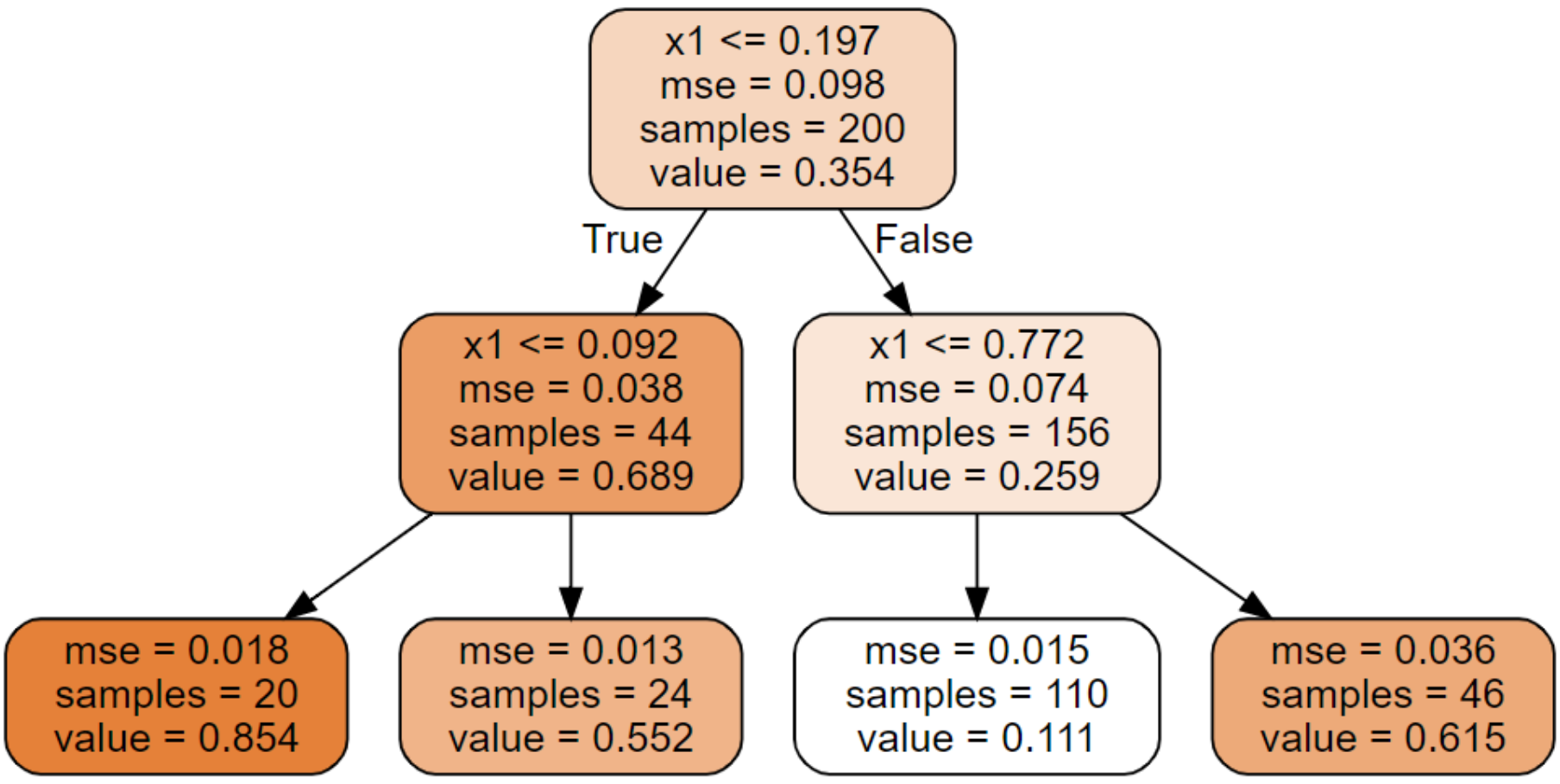
→ min_ 으로 시작하는 매개변수를 증가 시키거나 max_로 시작하는 매개변수를 감소 시키면 모델에 규제가 커진다.

- 제한없이 결정트리를 훈련시키고 불필요한 노드를 가지치기(제거) 하는 알고리즘도 있다. 순도를 높이는 것이 통계적으로 큰 효과가 없다면 리프노드 바로 위의 노드는 불필요할 수 있다. 대표적으로 카이제곱검정 같은 통계적 검정을 사용해 우연히 향상된 것인지 추정한다. 이 확률 값을 pvalue라고 부른다. 유의 수준보다 높으면 그 노드는 불필요한 것으로 간주되고 그 자식노드는 삭제된다. **가지치기**는 불필요한 노드가 모두 없어질 때 까지 계속된다.

📖 6.8 회귀

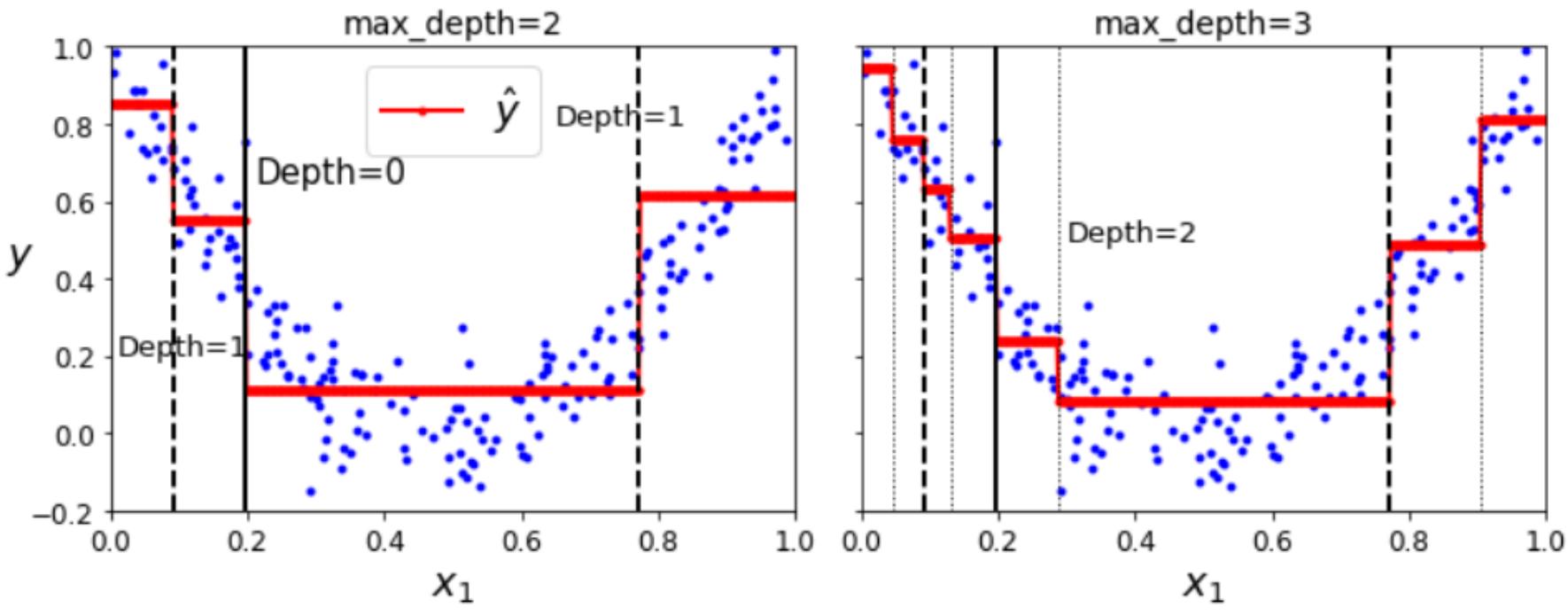
결정트리는 회귀 문제에도 사용할 수 있다. 분류와 결정적인 차이는, 각 노드에서 클래스를 예측하는 대신, 어떤 값을 예측한다는 것이다.

사이킷런을 사용해 잡음이 섞인 2차 함수 형태의 데이터 셋에서max_depth=2 설정으로 회귀 트리를 만들어 봄



→ **mse**를 추가적으로 계산 : 예를 들어 x1=0.6인 샘플의 타깃값을 예측해본다고 할 때, 루트노드부터 시작해 순회하여 결국 value=0.111인 리프노드에 도달한다. 이 리프노드에 있는 110개 훈련 샘플의 **평균 타깃값이 예측값**이 된다. 이 예측값을 사용해 110개 샘플에 대한 MSE를 구하면 0.015가 되는 것이다.

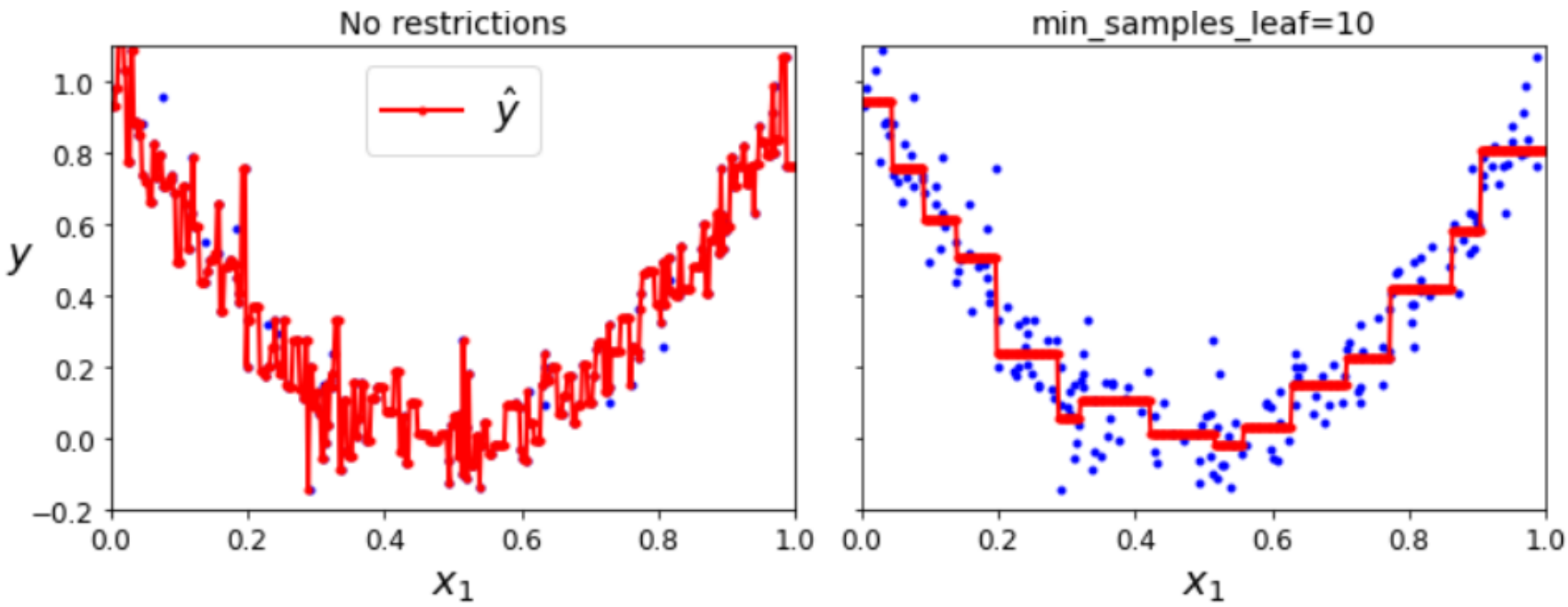
- 2개의 결정트리 회귀모델의 예측



→ 각 영역의 예측값은 항상 그 영역에 있는 타깃값의 평균이 된다. 알고리즘은 예측값과 가능한 많은 샘플이 가까이 있도록 영역을 분할한다.

- 결정트리 회귀 과정에서 CART 알고리즘은 MSE를 최소화 하도록 분할하는 것을 제외하고는 앞서 설명한 것과 거의 비슷하게 작동한다.

- **규제**

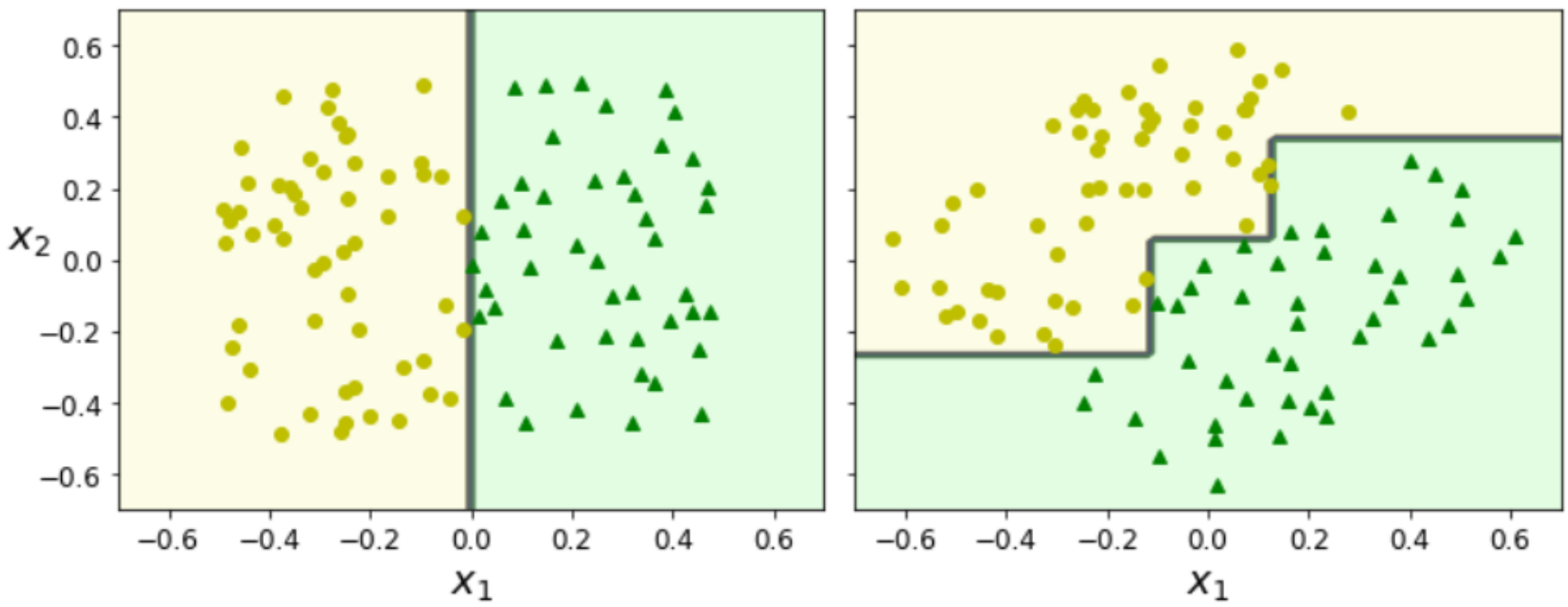


→ 분류에서와 같이 회귀 작업에서도 결정 트리가 과대적합되기 쉽다. 따라서 어느 정도 규제를 해도 좋다.

→ 왼쪽은 과대적합된 상황, 오른쪽은 `min_samples_leaf=10` 매개변수를 지정해 규제를 통해 어느정도 과대적합이 완화된 상황이다.

📖 6.9 불안전성

결정트리는 장점이 많지만, 계단 모양의 결정 경계를 만드므로(모든 분할은 축에 수직이다) 훈련세트의 회전에 민감하다.



→ 왼쪽의 결정트리는 쉽게 데이터셋을 구분하지만, 오른쪽의 데이터셋을 45도 회전한 결정트리는 불필요하게 구불구불 해졌다. 두 트리 모두 훈련세트를 완벽히 학습하지만 오른쪽 모델은 잘 일반화될 것 같진 않아 보인다. 이런 문제를 해결시키는 방법은 훈련 데이터를 더 좋은 방향으로 회전시키는 PCA 기법을 사용하는 것이다.

더불어 결정트리의 주된 문제는 훈련 데이터에 있는 작은 변화에도 매우 민감하다는 것이다. 이런 불안정성을 극복하는 방법은 다음 장에서 보게 될 랜덤 포레스트(많은 트리에서 만든 예측을 평균하는 방법)를 통해 배우게 된다!