

## 학습목표

- 성능이 좋고 이해하기 쉬운 트리 알고리즘에 대해 배웁니다.
- 알고리즘의 성능을 최대화하기 위한 하이퍼파라미터 튜닝을 실습합니다.
- 여러 트리를 합쳐 일반화 성능을 높일 수 있는 앙상블 모델을 배웁니다.

Chapter

# 05

## 트리 알고리즘

화이트 와인을 찾아라!

# 05-1

## 결정 트리

핵심 키워드

결정 트리

불순도

정보 이득

가지치기

특성 중요도

결정 트리 알고리즘을 사용해 새로운 분류 문제를 다루어 봅니다. 결정 트리가 머신러닝 문제를 어떻게 해결하는지 이해합니다.

“캔에 인쇄된 알코올 도수, 당도, pH 값으로 와인 종류를 구별할 수 있는 방법이 있을까?”

## 로지스틱 회귀로 와인 분류하기

혼공머신은 의외로 문제를 쉽게 풀 수 있을 것 같습니다. 품질관리 팀에서 6,497개의 와인 샘플 데이터를 보냈습니다. 이 데이터셋을 불러와 보죠. 4장에서처럼 판다스를 사용해 인터넷에서 직접 불러 오겠습니다. 다운로드할 주소는 [https://bit.ly/wine\\_csv\\_data](https://bit.ly/wine_csv_data)입니다.

### + 여기서 잠깐

#### 와인 데이터셋의 출처

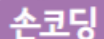
이 데이터셋은 캐글의 Red Wine Quality 데이터셋의 일부를 발췌한 것입니다.

- <https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009>

### 손코딩

```
import pandas as pd  
wine = pd.read_csv('https://bit.ly/wine_csv_data')
```

와인 데이터셋을 판다스 데이터프레임으로 제대로 읽어 들였는지 `head()` 메서드로 처음 5개의 샘플을 확인해 보겠습니다.

 손코딩

```
wine.head()
```



	alcohol	sugar	pH	class
0	9.4	1.9	3.51	0.0
1	9.8	2.6	3.20	0.0
2	9.8	2.3	3.26	0.0
3	9.8	1.9	3.16	0.0
4	9.4	1.9	3.51	0.0

처음 3개의 열(alcohol, sugar, pH)은 각각 알코올 도수, 당도, pH 값을 나타냅니다. 네 번째 열(class)은 타깃값으로 0이면 레드 와인, 1이면 화이트 와인이라고 하네요. 레드 와인과 화이트 와인을 구분하는 이진 분류 문제이고, 화이트 와인이 양성 클래스입니다. 즉 전체 와인 데이터에서 화이트 와인을 골라내는 문제군요.

손코딩

`wine.info()``<class 'pandas.core.frame.DataFrame'>``RangeIndex: 6497 entries, 0 to 6496``Data columns (total 4 columns):`

#	Column	Non-Null Count	Dtype
0	alcohol	6497 non-null	float64
1	sugar	6497 non-null	float64
2	pH	6497 non-null	float64
3	class	6497 non-null	float64

`dtypes: float64(4)``memory usage: 203.2 KB`

## Common Dtypes (Data Types)


Dtype	Meaning	Example Values
int64	64-bit integer (whole numbers)	1, 42, -100
float64	64-bit floating point (decimals)	3.14, 0.001, -2.5
object	Generic Python object (often strings)	'cat', 'yes'
bool	Boolean values	True, False
category	Categorical data (optimized)	'low', 'medium'

손코딩

`wine.describe()`


		alcohol	sugar	pH	class
	count	6497.000000	6497.000000	6497.000000	6497.000000
평균	mean	10.491801	5.443235	3.218501	0.753886
표준편차	std	1.192712	4.757804	0.160787	0.430779
최소	min	8.000000	0.600000	2.720000	0.000000
1사분위수	25%	9.500000	1.800000	3.110000	1.000000
중간값 / 2사분위수	50%	10.300000	3.000000	3.210000	1.000000
3사분위수	75%	11.300000	8.100000	3.320000	1.000000
최대	max	14.900000	65.800000	4.010000	1.000000

여기서 알 수 있는 것은 알코올 도수와 당도, pH 값의 스케일이 다르다는 것입니다. 이전에 했던 것처럼 사이킷런의 StandardScaler 클래스를 사용해 특성을 표준화해야겠군요. 그 전에 먼저 판다스 데이터프레임을 넘파이 배열로 바꾸고 훈련 세트와 테스트 세트로 나누겠습니다.

손코딩

```
data = wine[['alcohol', 'sugar', 'pH']].  
target = wine['class'].  
[redacted]
```


여기서 알 수 있는 것은 알코올 도수와 당도, pH 값의 스케일이 다르다는 것입니다. 이전에 했던 것처럼 사이킷런의 StandardScaler 클래스를 사용해 특성을 표준화해야겠군요. 그 전에 먼저 판다스 데이터프레임을 넘파이 배열로 바꾸고 훈련 세트와 테스트 세트로 나누겠습니다.


손코딩

```
data = wine[['alcohol', 'sugar', 'pH']].to_numpy()  
target = wine['class'].to_numpy()
```



wine 데이터프레임에서 처음 3개의 열을 넘파이 배열로 바꿔서 data 배열에 저장하고 마지막 class 열을 넘파이 배열로 바꿔서 target 배열에 저장했습니다. 이제 훈련 세트와 테스트 세트로 나누어 보죠.

손코딩

```
from sklearn.model_selection import train_test_split  
train_input, test_input, train_target, test_target =   
data, target, test_size=0.2, random_state=42)
```

wine 데이터프레임에서 처음 3개의 열을 넘파이 배열로 바꿔서 data 배열에 저장하고 마지막 class 열을 넘파이 배열로 바꿔서 target 배열에 저장했습니다. 이제 훈련 세트와 테스트 세트로 나누어 보죠.

 손코딩

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(
    data, target, test_size=0.2, random_state=42)
```


default value?




 손코딩

```
print(train_input.shape, test_input.shape)
```


```
(5197, 3) (1300, 3)
```

훈련 세트는 5,197개이고 테스트 세트는 1,300개입니다. 좋습니다. 이제 StandardScaler 클래스를 사용해 훈련 세트를 전처리해 보죠. 그다음 같은 객체를 그대로 사용해 테스트 세트를 변환하겠습니다.

손코딩


```
from sklearn.preprocessing import StandardScaler  
ss = StandardScaler()  
ss. (train_input)  
train_scaled = ss. (train_input)  
test_scaled = ss. (test_input)
```

훈련 세트는 5,197개이고 테스트 세트는 1,300개입니다. 좋습니다. 이제 StandardScaler 클래스를 사용해 훈련 세트를 전처리해 보죠. 그다음 같은 객체를 그대로 사용해 테스트 세트를 변환하겠습니다.

손코딩

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

모든 준비가 끝났네요. 이제 표준점수로 변환된 `train_scaled`와 `test_scaled`를 사용해 로지스틱 회귀 모델을 훈련하겠습니다.

손코딩

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_scaled, train_target)
print(lr.score(train_scaled, train_target))
print(lr.score(test_scaled, test_target))
```




0.7808350971714451  
0.7776923076923077

과대적합? 과소적합?

## 설명하기 쉬운 모델과 어려운 모델

혼공머신은 김 팀장과 함께 이사님에게 제출할 보고서를 만들려고 합니다. 이 모델을 설명하기 위해 로지스틱 회귀가 학습한 계수와 절편을 출력해 보죠.

 손코딩

```
print(lr.coef_, lr.intercept_)
```

```
[[ 0.51270274  1.6733911 -0.68767781]] [1.81777902]
```

---

자 이제 보고서를 작성해 봅시다.

### 보고서

작성자 : 혼공머신

이 모델은 알코올 도수 값에 0.51270274를 곱하고, 당도에 1.6733911을 곱하고, pH 값에  $-0.68767781$ 을 곱한 다음 모두 더합니다. 마지막으로 1.81777902를 더합니다. 이 값이 0보다 크면 화이트 와인, 작으면 레드 와인입니다. 현재 약 77% 정도를 정확히 화이트 와인으로 분류했습니다.  
모델의 출력 결과는...

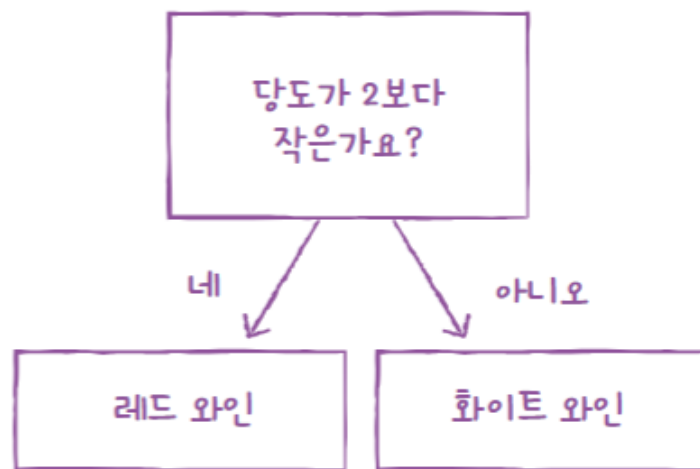


## 결정 트리

홍 선배는 혼공머신에게 **결정 트리** Decision Tree 모델이 “이유를 설명하기 쉽다”라고 알려 주었습니다. 생각해 보니 언뜻 책에서 본 것도 같네요. 결정 트리 모델은 스무고개와 같습니다. 다음의 오른쪽 그림처럼 질문을 하나씩 던져서 정답과 맞춰가는 거죠.

데이터를 잘 나눌 수 있는 질문을 찾는다면 계속 질문을 추가해서 분류 정확도를 높일 수 있습니다.

이미 예상했겠지만 사이킷런이 결정 트리 알고리즘을 제공합니다. 사이킷런의 DecisionTreeClassifier 클래스를 사용해 결정 트리 모델을 훈련해 보죠. 새로운 클래스이지만 사용법은 이전과 동일합니다. `fit()` 메서드를 호출해서 모델을 훈련한 다음 `score()` 메서드로 정확도를 평가해 보겠습니다.





- 의사결정 트리는 분류(Classification)와 회귀(Regression)를 위한 비모수적 지도 학습 방법
- 데이터의 특징(feature)으로부터 유추된 단순한 if-then-else 규칙을 통해 목표 변수(target variable)를 예측
- 트리는 구간별 상수 함수(piecewise constant function)로 볼 수 있으며, 트리의 깊이가 깊을수록 복잡하고 정밀한 예측이 가능

## 장점

- **이해와 해석이 쉬움**: 시각화가 가능하여 모델을 쉽게 설명할 수 있음.
- **데이터 전처리가 거의 필요 없음**: 정규화, 더미 변수 변환 없이도 사용 가능. 일부 알고리즘은 결측값도 처리 가능.
- **빠른 예측 속도**: 예측 비용이 로그( $\log n$ )에 비례.
- **수치형/범주형 데이터 모두 처리 가능**: (단, scikit-learn은 현재 범주형 직접 지원은 없음)
- **다중 출력 문제 해결 가능**: 예측 결과가 여러 개일 때도 사용 가능.
- **화이트박스 모델**: 각 예측의 근거를 논리적으로 설명할 수 있음.
- **통계적 검증 가능**: 모델의 신뢰도를 평가할 수 있는 방법 존재.
- **모델 가정에 덜 민감**: 현실 데이터의 복잡성에도 견고하게 작동 가능.

## 단점

- **과적합(Overfitting)** 위험: 너무 복잡한 트리를 만들면 일반화 성능이 떨어짐 → 가지치기(pruning), 최대 깊이 제한 등 필요.
- **데이터 민감성**: 입력 데이터에 약간의 변화만 있어도 완전히 다른 트리가 생성될 수 있음.
- **예측이 불연속적**: 예측값이 구간별 상수로 변화하기 때문에 매끄럽지 않고, 외삽(extrapolation)에 약함.
- **최적 트리 학습은 NP-완전 문제**: 전역 최적해를 보장하는 알고리즘은 현실적으로 어렵고, 대부분 탐욕적 방법(greedy algorithm)을 사용.
- **어려운 개념 학습에 한계**: XOR, 다중 선택기(MUX) 같은 논리 구조를 잘 표현하지 못함.
- **클래스 불균형에 취약**: 특정 클래스가 과도하게 많으면 편향된 트리를 생성할 수 있음 → 학습 전 데이터 균형 조정 필요.

손코딩

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(train_scaled, train_target)
print(dt.score(train_scaled, train_target)) # 훈련 세트
print(dt.score(test_scaled, test_target))  # 테스트 세트
```



0.996921300750433

0.8592307692307692

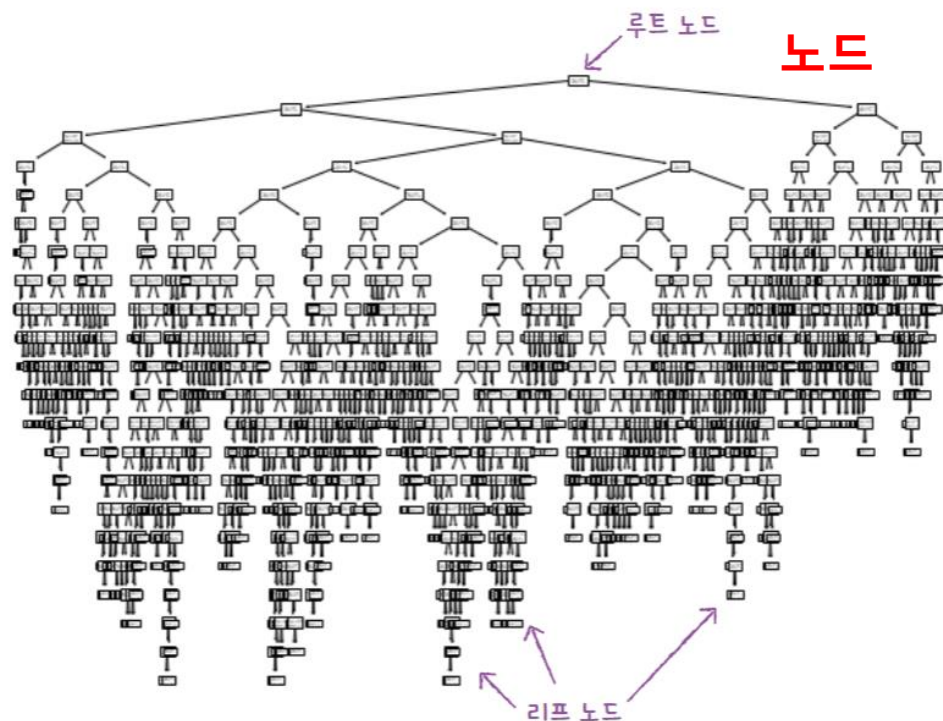
#### + 여기서 잠깐

#### 결정 트리 모델을 만들 때 왜 random\_state를 지정하나요?

사이킷런의 결정 트리 알고리즘은 노드에서 최적의 분할을 찾기 전에 특성의 순서를 섞습니다. 따라서 약간의 무작위성이 주입되는데 실행할 때마다 점수가 조금씩 달라질 수 있기 때문입니다. 여기에서는 독자들이 실습한 결과와 책의 내용이 같도록 유지하기 위해 random\_state를 지정하지만, 실전에서는 필요하지 않습니다.

손코딩

```
import matplotlib.pyplot as plt  
from sklearn.tree import plot_tree  
plt.figure(figsize=(10,7))  
plot_tree(dt)  
plt.show()
```



## 1. 루트 노드 (Root Node)

트리의 맨 위에 위치한 첫 번째 노드

전체 데이터에서 가장 중요한 기준(feature)을 사용하여 처음 분할을 수행

예: "나이 < 30?"

## 2. 내부 노드 (Internal Node / Decision Node)

데이터를 특정 기준(feature와 임계값)에 따라 둘 이상으로 나누는 노드

각각의 내부 노드는 조건문(if-else)을 포함하고 있음.

예: "수입 > 50,000?"

## 3. 잎 노드 (Leaf Node / Terminal Node)


더 이상 분할되지 않는 최종 노드로, 실제 예측값(또는 클래스 라벨)

각 잎 노드는 그 노드에 도달한 데이터 샘플들의 결과(출력)를 대표

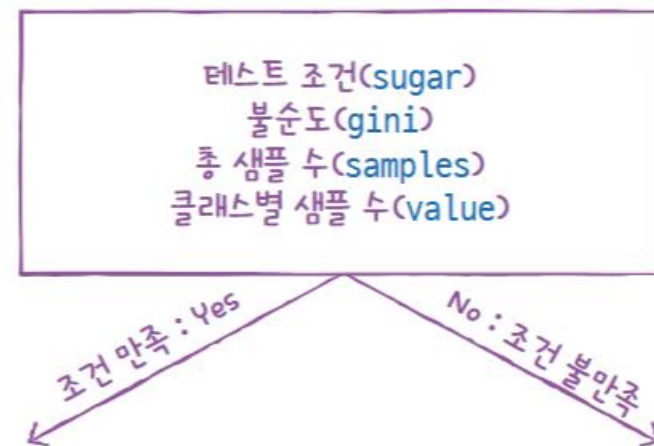
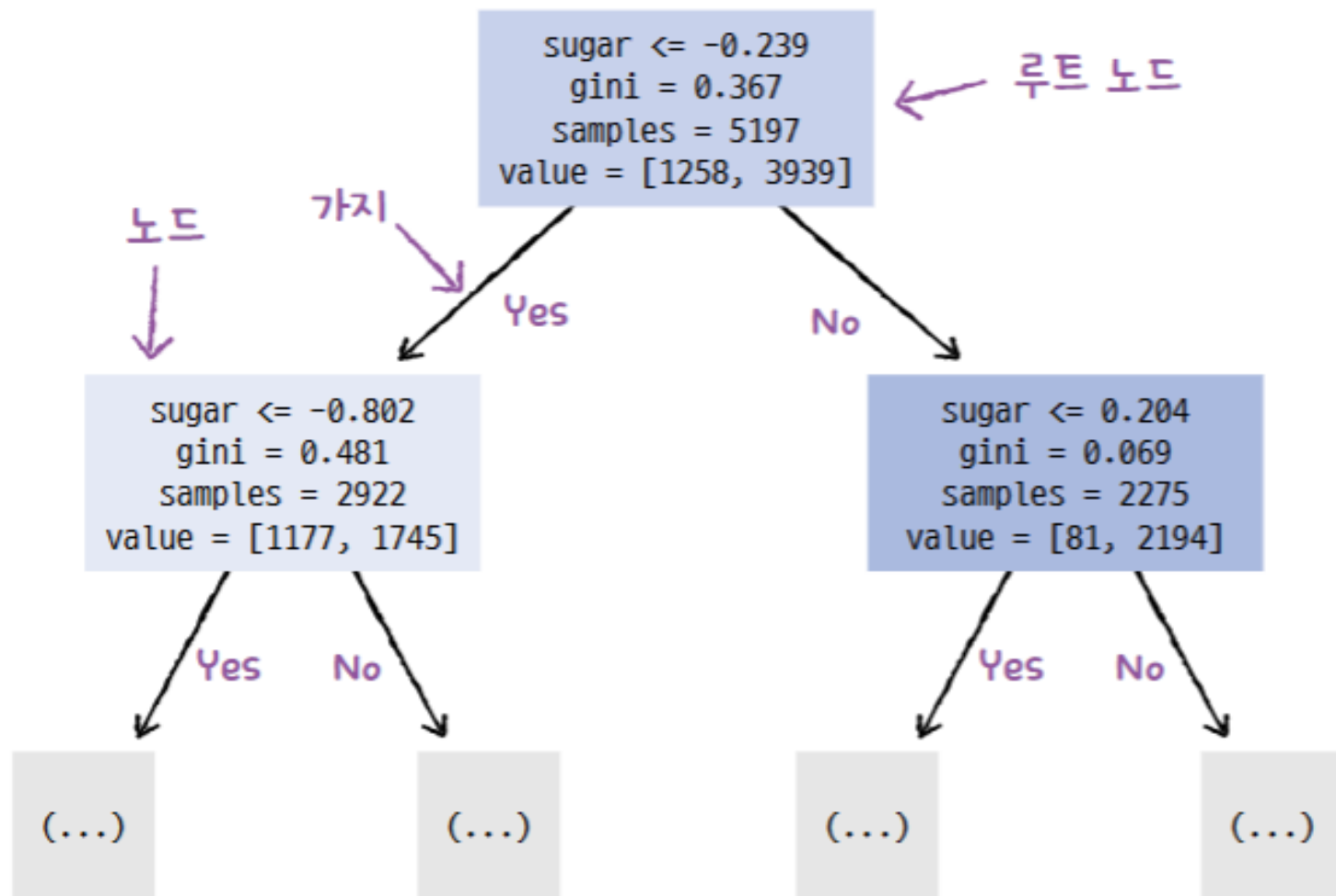
분류 문제에서는 클래스 라벨, 회귀 문제에서는 평균값이 주어짐.

예: "→ 클래스: Yes" 또는 "→ 예측값: 123.5"

너무 복잡하니 `plot_tree()` 함수에서 트리의 깊이를 제한해서 출력해 보죠. `max_depth` 매개변수를 1로 주면 루트 노드를 제외하고 하나의 노드를 더 확장하여 그립니다. 또 `filled` 매개변수에서 클래스에 맞게 노드의 색을 칠할 수 있습니다. `feature_names` 매개변수에는 특성의 이름을 전달할 수 있습니다. 이렇게 하면 노드가 어떤 특성으로 나뉘는지 좀 더 잘 이해할 수 있죠. 한번 이렇게 그려보겠습니다.

손코딩


```
plt.figure(figsize=(10,7))
plot_tree(dt, max_depth=1, filled=True, feature_names=['alcohol',
               'sugar', 'pH'])
plt.show()
```



**note** 만약 결정 트리를 회귀 문제에 적용하면 리프 노드에 도달한 샘플의 타겟을 평균하여 예측값으로 사용합니다. 사이킷런의 결정 트리 회귀 모델은 `DecisionTreeRegressor`입니다.

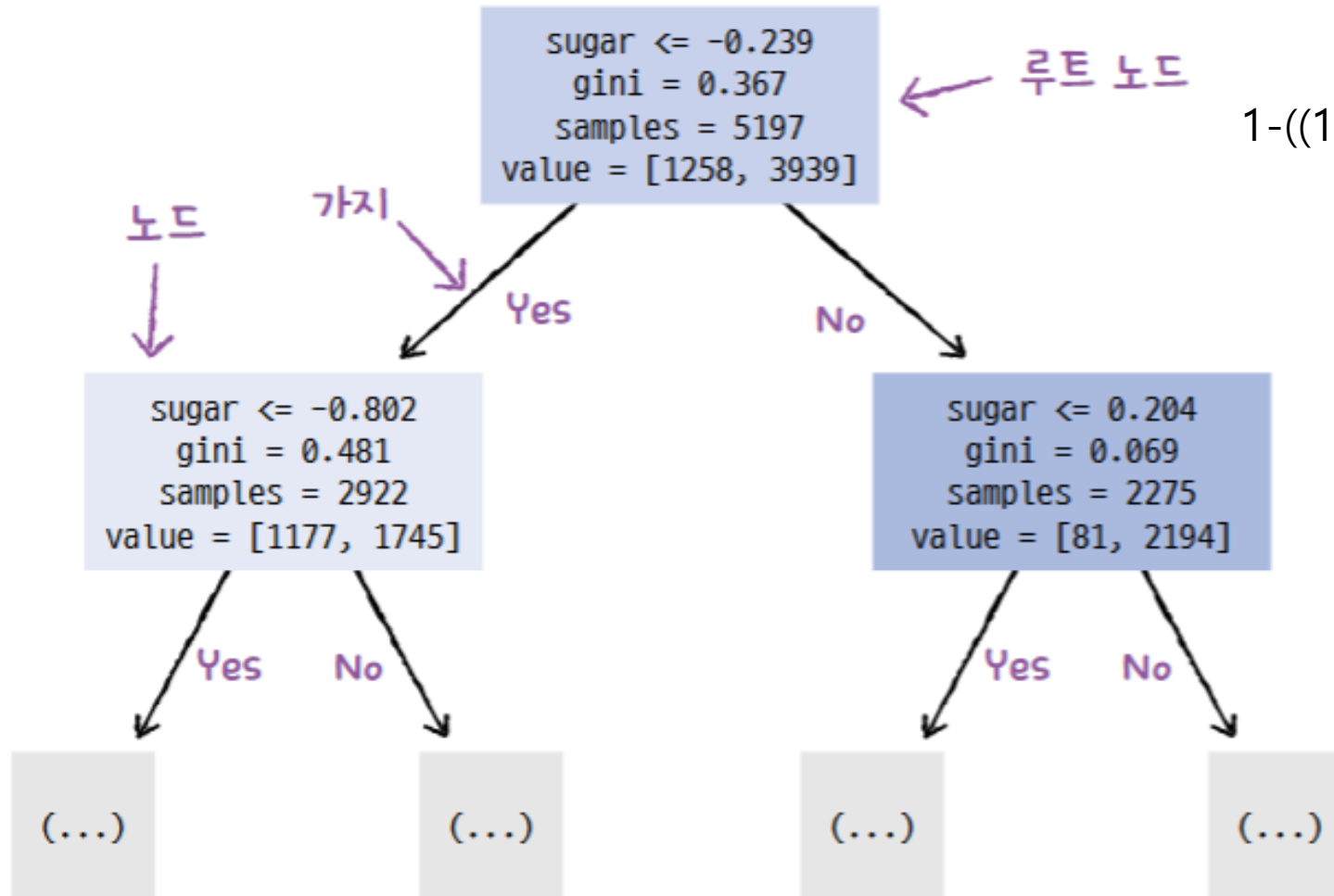


## 불순도

gini는 **지니 불순도**  Gini impurity를 의미합니다. DecisionTreeClassifier 클래스의 criterion 매개변수의 기본값이 'gini'입니다. criterion 매개변수의 용도는 노드에서 데이터를 분할할 기준을 정하는 것입니다. 앞의 **그린** 트리에서 루트 노드는 어떻게 당도 -0.239를 기준으로 왼쪽과 오른쪽 노드로 나누었을까요? 바로 criterion 매개변수에 지정한 지니 불순도를 사용합니다. 그럼 지니 불순도를 어떻게 계산하는지 알아보죠. 아주 간단합니다.

지니 불순도는 클래스의 비율을 제공해서 더한 다음 1에서 빼면 됩니다.

$$\text{지니 불순도} = 1 - (\text{음성 클래스 비율}^2 + \text{양성 클래스 비율}^2)$$



$$1 - ((1258/5197)^2 + (3939/5197)^2) = 0.367$$

결정 트리 모델은 부모 노드(parent node)와 자식 노드(child node)의 불순도 차이가 가능한 크도록 트리를 성장시킵니다. 부모 노드와 자식 노드의 불순도 차이를 계산하는 방법을 알아보죠. 먼저 자식 노드의 불순도를 샘플 개수에 비례하여 모두 더합니다. 그다음 부모 노드의 불순도에서 빼면 됩니다.

예를 들어 앞의 트리 그림에서 루트 노드를 부모 노드라 하면 왼쪽 노드와 오른쪽 노드가 자식 노드가 됩니다. 왼쪽 노드로는 2,922개의 샘플이 이동했고, 오른쪽 노드로는 2,275개의 샘플이 이동했습니다. 그럼 불순도의 차이는 다음과 같이 계산합니다.

$$\begin{aligned} & \text{부모의 불순도} - (\text{왼쪽 노드 샘플 수} / \text{부모의 샘플 수}) \times \text{왼쪽 노드 불순도} - \\ & (\text{오른쪽 노드 샘플 수} / \text{부모의 샘플 수}) \times \text{오른쪽 노드 불순도} = \\ & 0.367 - (2922 / 5197) \times 0.481 - (2275 / 5197) \times 0.069 = 0.066 \end{aligned}$$

정보이득 (information gain):  
부모 노드의 불순도와 자식 노드의 불순도의 차  
Maximize information gain!

- **Formula:**

$$Gini = 1 - \sum_{i=1}^n p_i^2$$

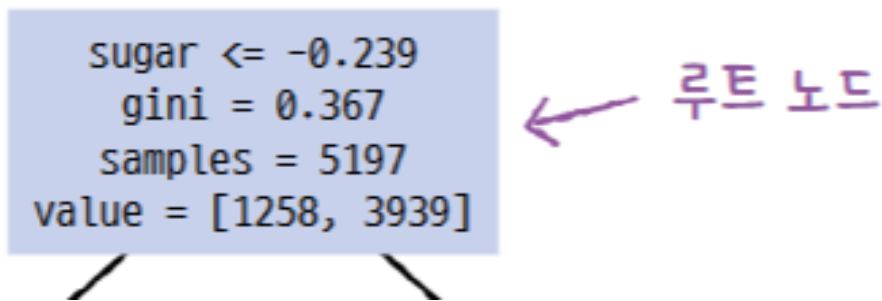
where  $p_i$  is the proportion of class  $i$  in the node.

- **Interpretation:**

- Gini is 0 when all elements belong to a single class (pure node).
- Higher Gini means more class mixing.
- Typically used in **CART (Classification and Regression Trees)**.
- **Faster to compute** than entropy, since it avoids logarithms.

DecisionTreeClassifier 클래스에서 criterion='entropy'를 지정하여 엔트로피 불순도를 사용할 수 있습니다. 엔트로피 불순도도 노드의 클래스 비율을 사용하지만 지니 불순도처럼 제공이 아니라 밑이 2인 로그를 사용하여 곱합니다. 예를 들어 루트 노드의 엔트로피 불순도는 다음과 같이 계산할 수 있습니다.

$$\begin{aligned} & -\text{음성 클래스 비율} \times \log_2(\text{음성 클래스 비율}) - \text{양성 클래스 비율} \times \log_2(\text{양성 클래스 비율}) \\ &= -(1258 / 5197) \times \log_2(1258 / 5197) - \\ & (3939 / 5197) \times \log_2(3939 / 5197) = 0.798 \end{aligned}$$



- **Formula:**

$$Entropy = - \sum_{i=1}^n p_i \log_2(p_i)$$

- **Interpretation:**

- Entropy is 0 for a pure node (just like Gini).
  - Maximum entropy occurs when classes are equally mixed.
  - Used in algorithms like **ID3**, **C4.5**, and **C5.0**.
- When splitting a node, **Information Gain** is:


$$IG = Entropy(parent) - \sum_k \left( \frac{|child_k|}{|parent|} \cdot Entropy(child_k) \right)$$

**Not for test but good to know!**

Algorithm	Split Criterion	Handles Continuous	Pruning	Missing Values	Notes
<b>ID3</b>	Information Gain	✗	✗	✗	Basic, overfits easily
<b>C4.5</b>	Gain Ratio	✓	✓	✓	Strong general-purpose classifier
<b>C5.0</b>	Boosted Gain Ratio	✓	✓	✓	Faster, more accurate (commercial)

## 가지치기

그럼 가지치기를 해 보죠. 결정 트리에서 가지치기를 하는 가장 간단한 방법은 자라날 수 있는 트리의 최대 깊이를 지정하는 것입니다. DecisionTreeClassifier 클래스의 max\_depth 매개변수를 3으로 지정하여 모델을 만들어 보겠습니다. 이렇게 하면 루트 노드 아래로 최대 3개의 노드까지만 성장할 수 있습니다.

 손코딩


```
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(train_scaled, train_target)
print(dt.score(train_scaled, train_target))
print(dt.score(test_scaled, test_target))
```



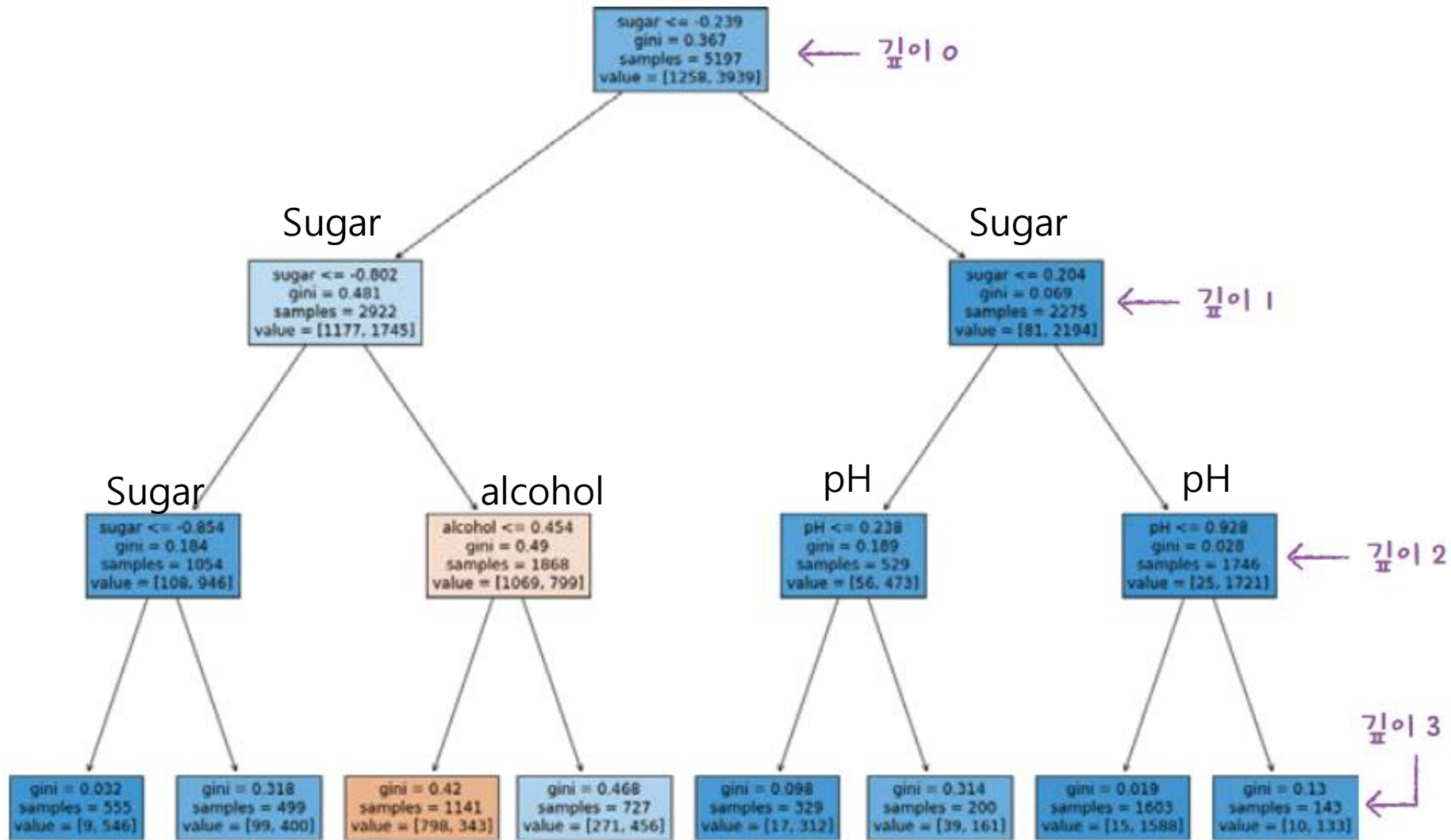
```
0.8454877814123533
0.8415384615384616
```




훈련 세트의 성능은 낮아졌지만 테스트 세트의 성능은 거의 그대로입니다. 이런 모델을 트리 그래프로 그린다면 훨씬 이해하기 쉬울 것 같네요. `plot_tree()` 함수로 그려 보죠.

 손코딩

```
plt.figure(figsize=(20,15))  
plot_tree(dt, filled=True, feature_names=['alcohol', 'sugar', 'pH'])  
plt.show()
```



그런데  $-0.802$ 라는 음수로 된 당도를 이사님께 어떻게 설명해야 할까요? 잠깐만요. 뭔가 이상하군요. 앞서 불순도를 기준으로 샘플을 나누다고 했습니다. 불순도는 클래스별 비율을 가지고 계산했죠. 샘플을 어떤 클래스 비율로 나누는지 계산할 때 특성값의 스케일이 계산에 영향을 미칠까요? 아니요, 특성값의 스케일은 결정 트리 알고리즘에 아무런 영향을 미치지 않습니다. 따라서 표준화 전처리를 할 필요가 없습니다. 이것이 결정 트리 알고리즘의 또 다른 장점 중 하나입니다.



결정 트리는 표준화 전처리 과정이 필요 없습니다.

그럼 앞서 전처리하기 전의 훈련 세트(train\_input)와 테스트 세트(test\_input)로 결정 트리 모델을 다시 훈련해 보겠습니다.

손코딩

```
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(train_input, train_target)
print(dt.score(train_input, train_target))
print(dt.score(test_input, test_target))
```

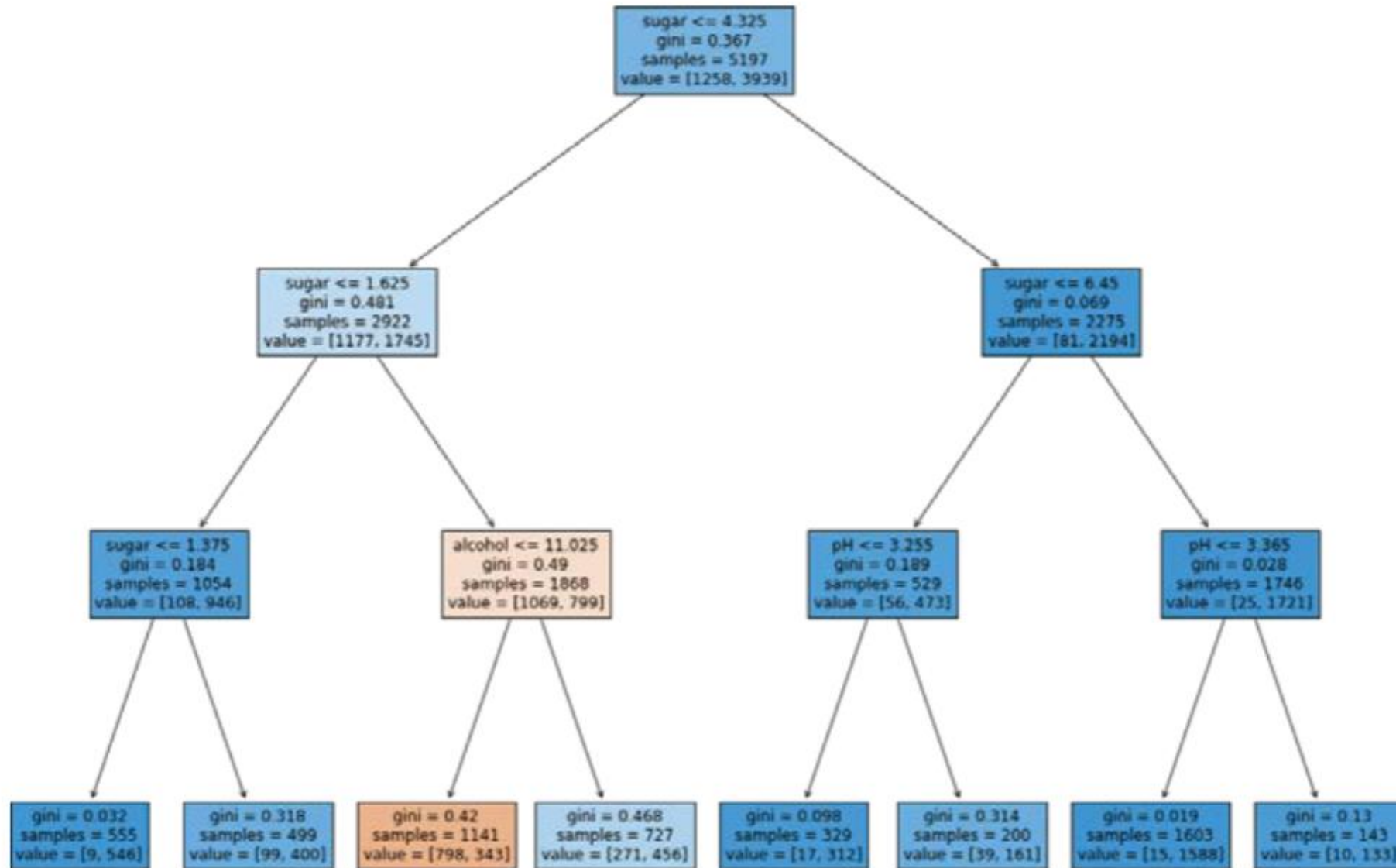


```
0.8454877814123533
0.8415384615384616
```


---

손코딩

```
plt.figure(figsize=(20,15))
plot_tree(dt, filled=True, feature_names=['alcohol', 'sugar', 'pH'])
plt.show()
```



마지막으로 결정 트리는 어떤 특성이 가장 유용한지 나타내는 특성 중요도를 계산해 줍니다. 이 트리의 루트 노드와 깊이 1에서 당도를 사용했기 때문에 아마도 당도(sugar)가 가장 유용한 특성 중 하나일 것 같습니다. 특성 중요도는 결정 트리 모델의 `feature_importances_` 속성에 저장되어 있습니다. 이 값을 출력해 확인해 보죠.

 손코딩

```
print(dt.feature_importances_)
```

```
[0.12345626 0.86862934 0.0079144 ]
```

네, 역시 두 번째 특성인 당도가 0.87 정도로 특성 중요도가 가장 높네요. 그다음 알코올 도수, pH 순입니다. 이 값을 모두 더하면 1이 됩니다. 특성 중요도는 각 노드의 정보 이득과 전체 샘플에 대한 비율을 곱한 후 특성별로 더하여 계산합니다. 특성 중요도를 활용하면 결정 트리 모델을 특성 선택에 활용할 수 있습니다. 이것이 결정 트리 알고리즘의 또 다른 장점 중 하나입니다.

결정 트리의 특성 중요도를 특성 선택에 활용할 수 있습니다.

## ▶ 키워드로 끝내는 핵심 포인트

- **결정 트리**는 예 / 아니오에 대한 질문을 이어나가면서 정답을 찾아 학습하는 알고리즘입니다. 비교적 예측 과정을 이해하기 쉽고 성능도 뛰어납니다.
- **불순도**는 결정 트리가 최적의 질문을 찾기 위한 기준입니다. 사이킷런은 지니 불순도와 엔트로피 불순도를 제공합니다.
- **정보 이득**은 부모 노드와 자식 노드의 불순도 차이입니다. 결정 트리 알고리즘은 정보 이득이 최대화되도록 학습합니다.
- 결정 트리는 제한 없이 성장하면 훈련 세트에 과대적합되기 쉽습니다. **가지치기**는 결정 트리의 성장을 제한하는 방법입니다. 사이킷런의 결정 트리 알고리즘은 여러 가지 가지치기 매개변수를 제공합니다.
- **특성 중요도**는 결정 트리에 사용된 특성이 불순도를 감소하는데 기여한 정도를 나타내는 값입니다. 특성 중요도를 계산할 수 있는 것이 결정 트리의 또다른 큰 장점입니다.



## scikit-learn

- **DecisionTreeClassifier**는 결정 트리 분류 클래스입니다.

`criterion` 매개변수는 불순도를 지정하며 기본값은 지니 불순도를 의미하는 'gini'이고 'entropy'를 선택하여 엔트로피 불순도를 사용할 수 있습니다.

`splitter` 매개변수는 노드를 분할하는 전략을 선택합니다. 기본값은 'best'로 정보 이득이 최대가 되도록 분할합니다. 'random'이면 임의로 노드를 분할합니다.

`max_depth`는 트리가 성장할 최대 깊이를 지정합니다. 기본값은 None으로 리프 노드가 순수하거나 `min_samples_split`보다 샘플 개수가 적을 때까지 성장합니다.

`min_samples_split`은 노드를 나누기 위한 최소 샘플 개수입니다. 기본값은 2입니다.

`max_features` 매개변수는 최적의 분할을 위해 탐색할 특성의 개수를 지정합니다. 기본값은 None으로 모든 특성을 사용합니다.

- **plot\_tree()**는 결정 트리 모델을 시각화합니다. 첫 번째 매개변수로 결정 트리 모델 객체를 전달합니다.

`max_depth` 매개변수로 나타낼 트리의 깊이를 지정합니다. 기본값은 None으로 모든 노드를 출력합니다.

`feature_names` 매개변수로 특성의 이름을 지정할 수 있습니다.

`filled` 매개변수를 True로 지정하면 타깃값에 따라 노드 안에 색을 채웁니다.



## ▶ 확인 문제

05-1

결정 트리

1. 다음 중 결정 트리의 불순도에 대해 옳게 설명한 것을 모두 고르세요.

- ① 지니 불순도는 부모 노드의 불순도와 자식 노드의 불순도의 차이로 계산합니다.
- ② 지니 불순도는 클래스의 비율을 제공하여 모두 더한 다음 1에서 뺍니다.
- ③ 엔트로피 불순도는 1에서 가장 큰 클래스 비율을 빼서 계산합니다.
- ④ 엔트로피 불순도는 클래스 비율과 클래스 비율에 밑이 2인 로그를 적용한 값을 곱해서 모두 더한 후 음수로 바꾸어 계산합니다.

지니 불순도 계산식 :  $1 - (\text{양성 클래스 비율}^2 + \text{음성 클래스 비율}^2)$

엔트로피 불순도 계산식 :  $-(\text{음성 클래스 비율} \times \log_2(\text{음성 클래스 비율}) + \text{양성 클래스 비율} \times \log_2(\text{양성 클래스 비율}))$

2. 결정 트리에서 계산한 특성 중요도가 저장되어 있는 속성은 무엇인가요?

- ① important\_variables\_
- ② variable\_importances\_
- ③ important\_features\_
- ④ feature\_importances\_

3. 앞서 결정 트리 예제에서 max\_depth를 3으로 지정하여 좌우가 대칭인 트리를 만들었습니다. 사이킷런의 결정 트리 클래스가 제공하는 매개변수 중 min\_impurity\_decrease를 사용해 가지치기를 해 보겠습니다. 어떤 노드의 정보 이득  $\times$  (노드의 샘플 수) / (전체 샘플 수) 값이 이 매개변수보다 작으면 더 이상 분할하지 않습니다. 이 매개변수의 값을 0.0005로 지정하고 결정 트리를 만들어 보세요. 좌우가 균일하지 않은 트리가 만들어지나요? 테스트 세트의 성능은 어떤가요?

```
dt = DecisionTreeClassifier( , random_state=42) # 코드를 완성해 보세요
dt.fit(train_input, train_target)
print(dt.score(train_input, train_target))
print(dt.score(test_input, test_target))
plt.figure(figsize=(20,15))
plot_tree(dt, filled=True, feature_names=['alcohol', 'sugar', 'pH'])
plt.show()
```

