CHAPTER 05 트리 알고리즘

# 인공지능

대전대학교 컴퓨터공학과

조교수 박상돈

# 학습 로드맵

**머신러닝편** **01~06장**

딥러닝만 먼저 배우고
싶다면 01~04장을 읽은 후
07장으로 건너뛰어도 좋습니다.

**딥러닝편** **07~10장**

07장을 읽은 후 08장과 09장은
순서대로 읽지 않아도 괜찮습니다.
10장을 읽기 전에 07장과 09장을
읽는 것이 좋습니다.

**난이도** ●●●●●

START

01 나의 첫 머신러닝 ●○○○○

02 데이터 다루기 ●●○○○

03 회귀 알고리즘과 모델 규제

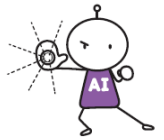
2번 보기

04 다양한 분류 알고리즘 ●●○○○

05 트리 알고리즘 ●●●○○

06 비지도 학습 ●●●○○

07 딥러닝을 시작합니다 ●●●●○




08 이미지를 위한 인공 신경망 ●●●●●

09 텍스트를 위한 인공 신경망 ●●●●●

10 언어 모델을 위한 신경망 ●●●●●

GOAL

# Contents

## CHAPTER 05 트리 알고리즘

# CHAPTER 05 트리 알고리즘

## 화이트 와인을 찾아라!

**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.

# SECTION 5-1 결정 트리(1)

- 캔에 인쇄된 알코올 도수, 당도, pH 값으로 레드 와인과 화이트 와인 구별하기
- 로지스틱 회귀로 와인 분류하기
  - 6,497개의 와인 샘플 데이터 가져오기 (소스 https://bit.ly/wine-date)

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

  - head( ) 메서드로 처음 5개의 샘플을 확인

```
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
네 번째 열(class )은 타깃값. 0이면 레드 와인, 1이면 화이트 와인
레드 와인과 화이트 와인을 구분하는 이진 분류 문제이고,
화이트 와인이 양성 클래스

# SECTION 5-1 결정 트리(2)

◦ 로지스틱 회귀로 와인 분류하기

- info( ) 메서드: 데이터프레임의 각 열의 데이터 타입과 누락된 데이터가 있는지 확인

```
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
```

▲ 총 6,497개의 샘플이 있고 4개의 열은 모두 실숫값
Non-Null Count가 모두 6497이므로 누락된 값은 없음

# SECTION 5-1 결정 트리(3)

◦ 로지스틱 회귀로 와인 분류하기

- describe( ) 매사드: 열에 대한 간략한 통계(최소, 최대, 평균값 등)를 출력

```
wine.describe()
```

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

▲ 도수, 당도, pH 스케일이 다름

StandardScaler 클래스를 사용해 특성을 표준화 필요

# SECTION 5-1 결정 트리(4)

- 로지스틱 회귀로 와인 분류하기
  - 판다스 데이터프레임을 훈련 세트와 테스트 세트로 나누기

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

    - [노트]실습과 결괏값이 같도록 random_state를 42로 고정

```
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)
```

  - 훈련 세트와 테스트 세트의 크기 확인

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

→ (5197, 3) (1300, 3)

  - StandardScaler 클래스를 사용해 훈련 세트 전처리와 테스트 세트 변환

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

# SECTION 5-1 결정 트리(5)

◦ 로지스틱 회귀로 와인 분류하기

- 표준점수로 변환된 train_scaled와 test_scaled를 사용해 로지스틱 회귀 모델을 훈련

```
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

▲ 훈련 세트와 테스트 세트의 점수가 모두 낮아 다소 과소적합

# SECTION 5-1 결정 트리(6)

- 로지스틱 회귀로 와인 분류하기
  - 설명하기 쉬운 모델과 어려운 모델
    - 모델을 설명하기 위한 보고서 작성을 위해, 로지스틱 회귀가 학습한 계수와 절편을 출력

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

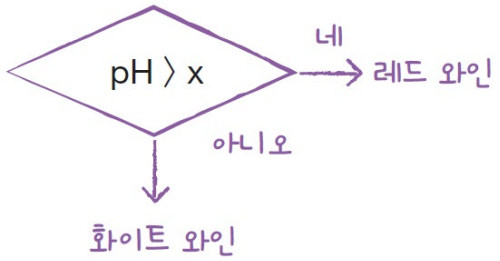
→ [[ 0.51268071  1.67335441 -0.68775646]] [1.81773456]

보고서

작성자 : 혼공머신

이 모델은 알코올 도수 값에 0.51268071를 곱하고, 당도에 1.67335441을 곱하고, pH 값에 −0.68775646을 곱한 다음 모두 더합니다. 마지막으로 1.81773456를 더합니다. 이 값이 0보다 크면 화이트 와인, 작으면 레드 와인입니다. 현재 약 78% 정도를 정확히 화이트 와인으로 분류했습니다.
모델의 출력 결과는...

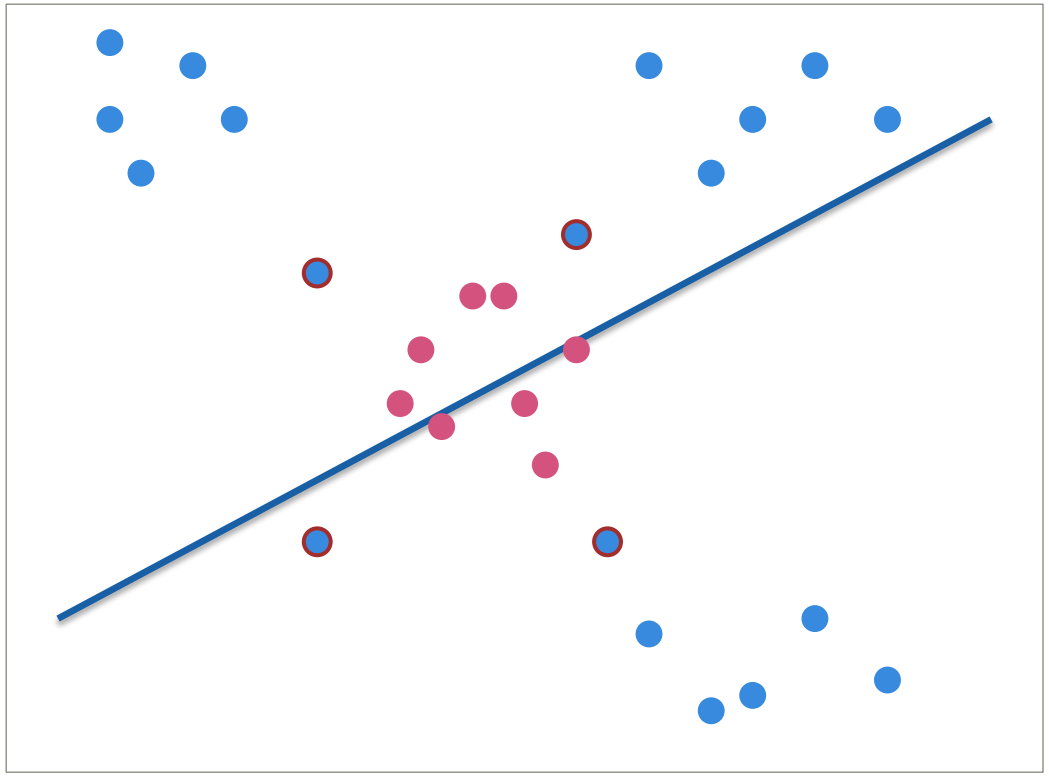

- 순서도로 표현?



# SECTION 5-1 결정 트리의 핵심 직관

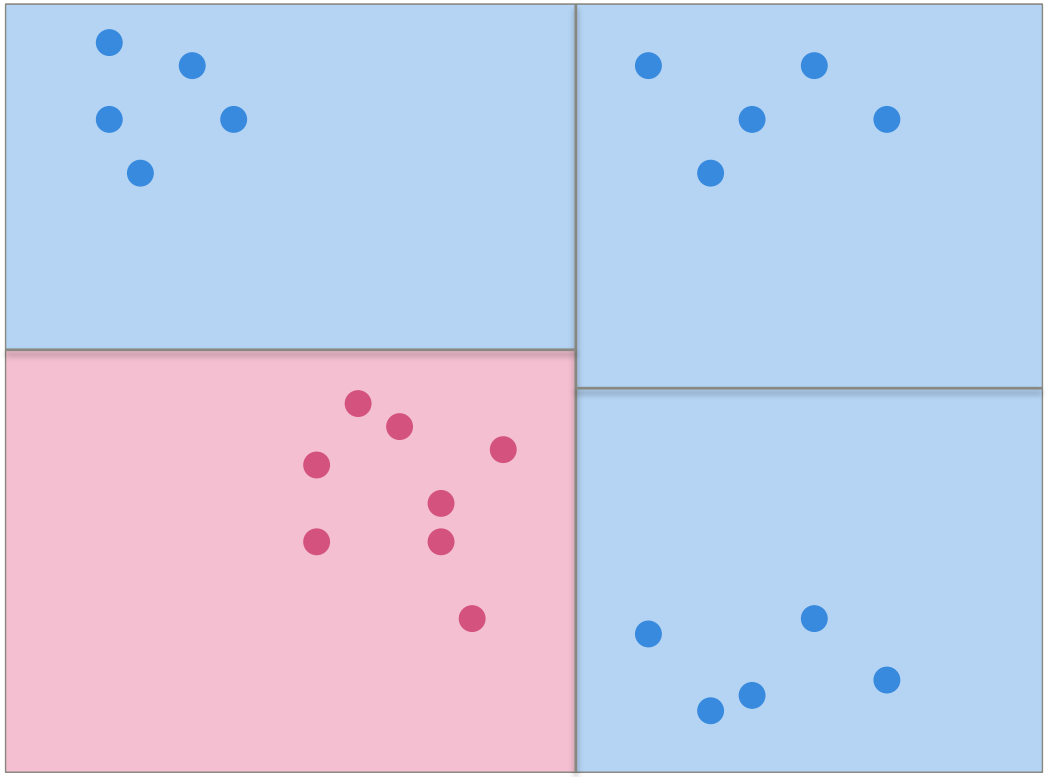
직선 한 개로 자르기 vs 공간을 칸으로 쪼개기

### 로지스틱 회귀: 직선 하나로 자르기



약 78% – 직선 한 개로는 흩어진 화이트 와인을 다 못 잡음

### 결정 트리: 공간을 여러 칸으로 쪼개기



약 95% – 칸을 잘 쪼개면 모든 영역을 표현 가능

**핵심 차이: 로지스틱 회귀의 표현력은 "직선 한 개"가 한계 – 결정 트리는 칸을 늘리면 어떤 모양이든 근사 가능**

# SECTION 5-1 결정 트리(7)

◦ 결정 트리(Decision Tree)

▪ 사이킷런의 DecisionTreeClassifier 클래스를 사용해 결정 트리 모델을 훈련

• fit( ) 메서드를 호출해서 모델을 훈련한 다음 score( ) 메서드로 정확도를 평가

```
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
```

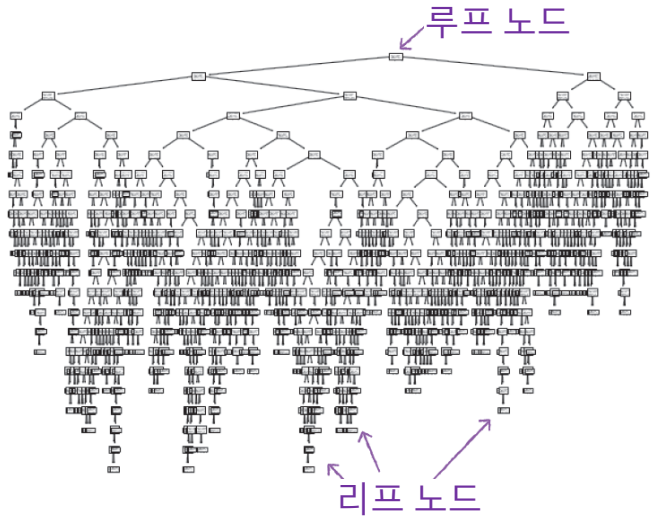
→

```
0.996921300750433
0.8592307692307692
```

• plot_tree( ) 함수를 사용해 결정 트리를 이해하기 쉬운 트리 그림으로 출력

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

→

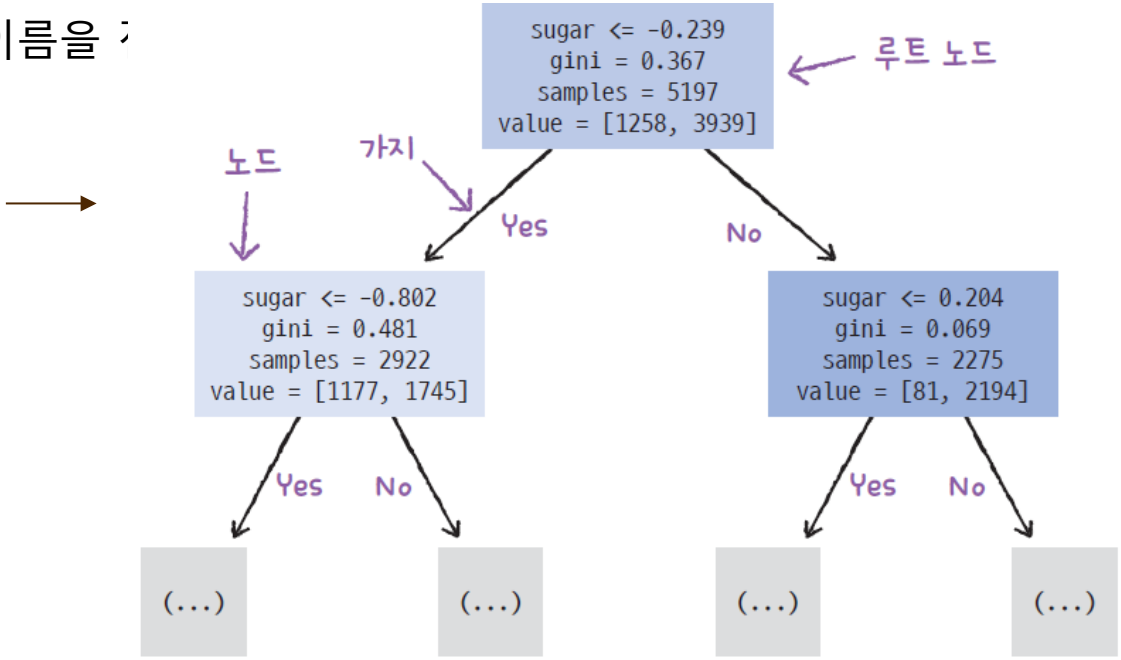


# SECTION 5-1 결정 트리(8)

◦ 결정 트리(Decision Tree)

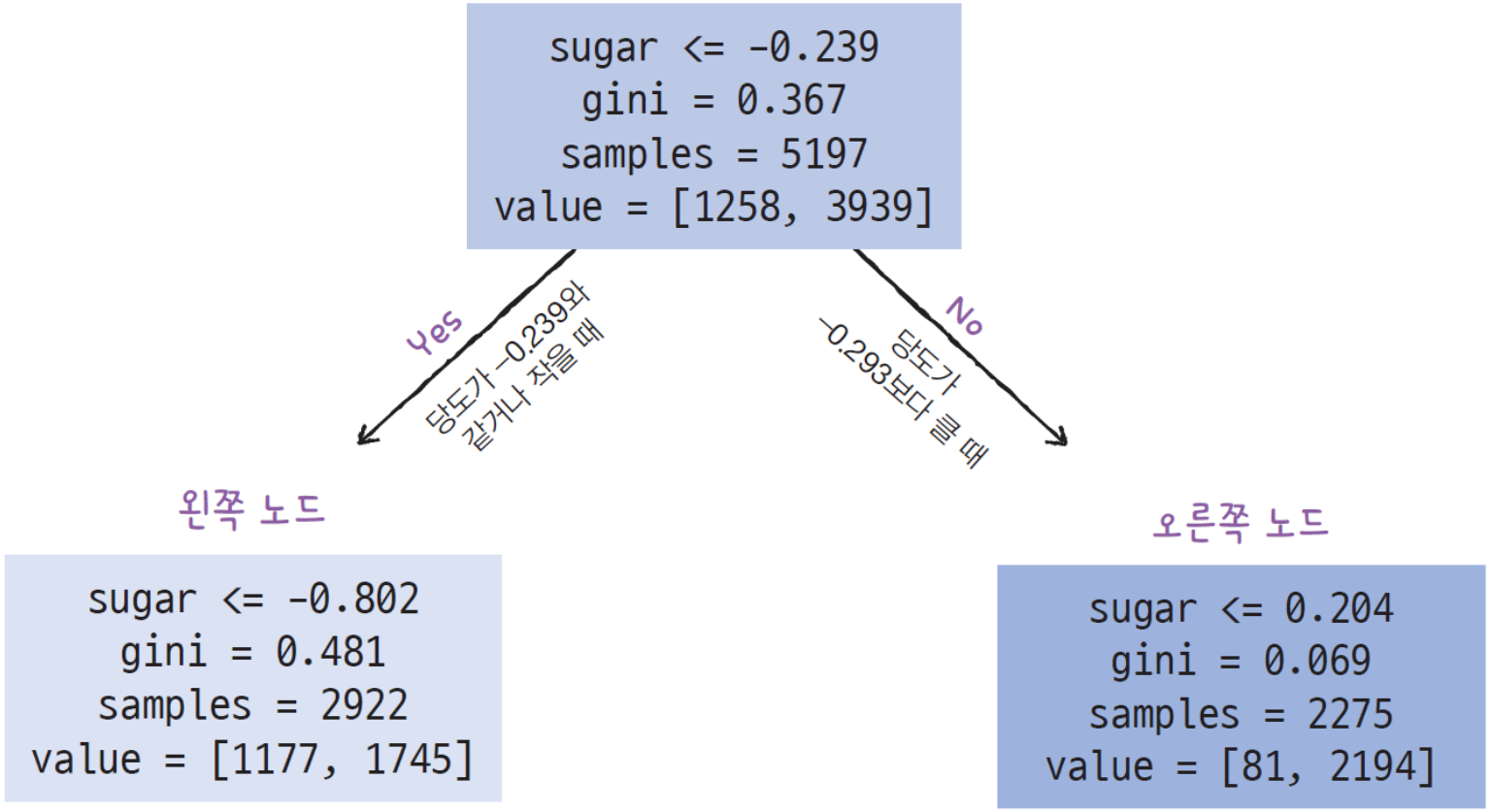
- 사이킷런의 DecisionTreeClassifier 클래스를 사용해 결정 트리 모델을 훈련
  - plot_tree( ) 함수에서 트리의 깊이를 제한해서 출력
  - max_depth 매개변수를 1로 주면 루트 노드를 제외하고 하나의 노드를 더 확장
  - filled 매개변수에서 클래스에 맞게 노드의 색깔 조정
  - feature_names 매개변수에는 특성의 이름을 ?

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



# SECTION 5-1 결정 트리(9)

- 결정 트리(Decision Tree)
  - 사이킷런의 DecisionTreeClassifier 클래스를 사용해 결정 트리 모델을 훈련



# SECTION 5-1 결정 트리(10)

- 결정 트리(Decision Tree)
  - 불순도(impurity)
    - gini는 지니 불순도(Gini impurity)를 의미
    - DecisionTreeClassifier 클래스의 criterion 매개변수의 기본값이 'gini'
    - criterion 매개변수: 노드에서 데이터를 분할할 기준을 정함
    - 앞의 그린 트리에서 루트 노드는 당도 -0.239를 기준으로 왼쪽과 오른쪽 노드로 나눌 때, criterion 매개변수에 지정한 지니 불순도를 사용

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

    - 루트 노드는 총5 ,197개의 샘플, 그중에 1,258개가 음성 클래스, 3,939개가 양성 클래스인 경우의 지니불순도

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

    - 순수 노드: 노드에 하나의 클래스만 있어, 지니 불순도가 0인 노드

$$1 - ((0 / 100)^2 + (100 / 100)^2) = 0$$

# SECTION 5-1 결정 트리(11)

- 결정 트리(Decision Tree)
  - 불순도(impurity)
    - 결정 트리 모델은 부모 노드(parent node)와 자식 노드(child node)의 불순도 차이가 가능한 크도록 트리를 성장시킴
    - 정보 이득(information gain): 부모 노드와 자식 노드의 불순도 차이

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

    - 엔트로피 불순도: DecisionTreeClassifier 클래스에서 criterion='entropy'를 지정하여 사용

$$\textit{음성 클래스 비율} \times \log_2(\textit{음성 클래스 비율}) - \textit{양성 클래스 비율} \times \log_2(\textit{양성 클래스 비율})$$
$$= -(1258 / 5197) \times \log_2(1258 / 5197) -$$
$$(3939 / 5197) \times \log_2(3939 / 5197) = 0.798$$

    - 결정 트리 알고리즘
      - 불순도 기준을 사용해 정보 이득이 최대가 되도록 노드를 분할
      - 노드를 순수하게 나눌수록 정보 이득이 커짐
      - 새로운 샘플에 대해 예측할 때에는 노드의 질문에 따라 트리를 이동
      - 마지막에 도달한 노드의 클래스 비율을 보고 예측

# SECTION 5-1 결정 트리(12)

- 결정 트리(Decision Tree)
  - 가지치기
    - 결정 트리에서 자라날 수 있는 트리의 최대 깊이를 지정
    - DecisionTreeClassifier 클래스의 max_depth 매개변수를 3으로 지정하여 모델 만들기

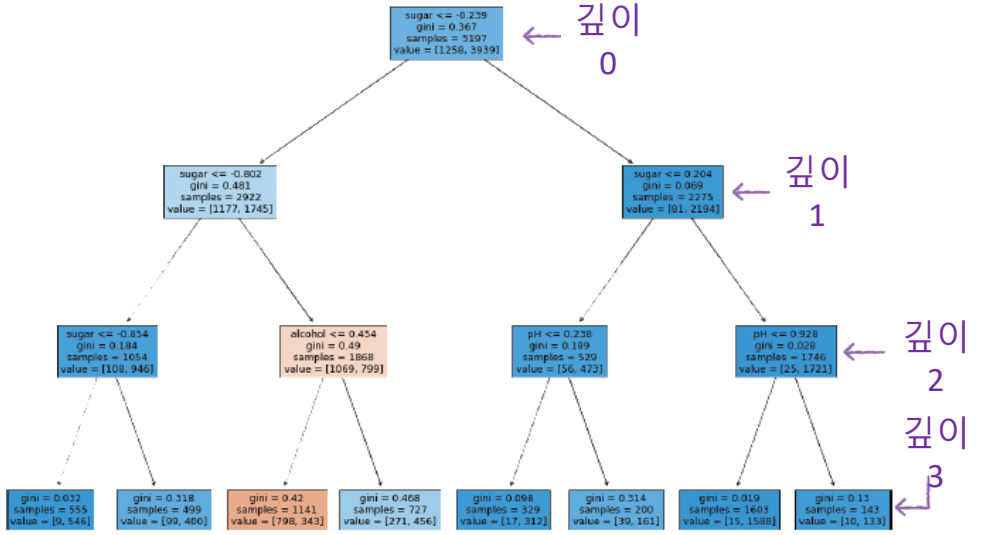
```
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(train_scaled, train_target)
print(dt.score(train_scaled, train_target))
print(dt.score(test_scaled, test_target))
```

→ 0.8454877814123533
0.8415384615384616

    - plot_tree( ) 함수로 트리 그래프 작성

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

→



# SECTION 5-1 결정 트리(13)

◦ 결정 트리(Decision Tree)

- 가지치기

  - 특성값의 스케일은 결정 트리 알고리즘에 아무런 영향을 미치지 않으므로 표준화 전처리를 할 필요가 없음
  - 전처리하기 전의 훈련 세트(train_input )와 테스트 세트(test_input )로 결정 트리 모델을 다시 훈련

```
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))
```

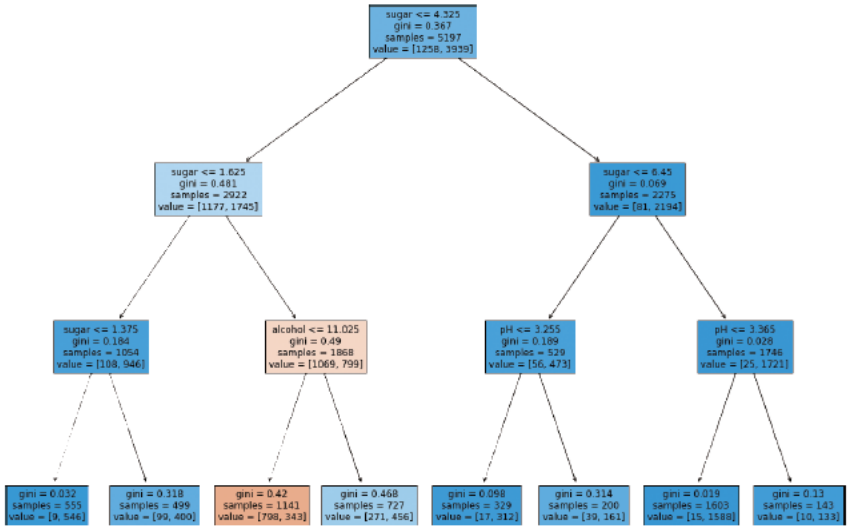
→

0.8454877814123533
0.8415384615384616

  - 트리 그래프로 구현

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

→



# SECTION 5-1 결정 트리(14)

◦ 결정 트리(Decision Tree)

- 가지치기

  - 결정 트리는 어떤 특성이 가장 유용한지 나타내는 특성 중요도를 계산
  - 앞의 트리에서는 루트 노드와 깊이 1에서 당도를 사용했기 때문에 아마도 당도(sugar)가 가장 유용한 특성 중 하나일 것임
  - 특성 중요도는 결정 트리 모델의 feature_importances_ 속성에 저장됨. 이 값을 출력해 확인

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

→ [0.12345626 0.86862934 0.0079144 ]

▲ 두 번째 특성인 당도가 0.87 정도로 특성 중요도가 가장 높고, 그다음 알코올 도수, pH 순. 이 값을 모두 더하면 1이 됨

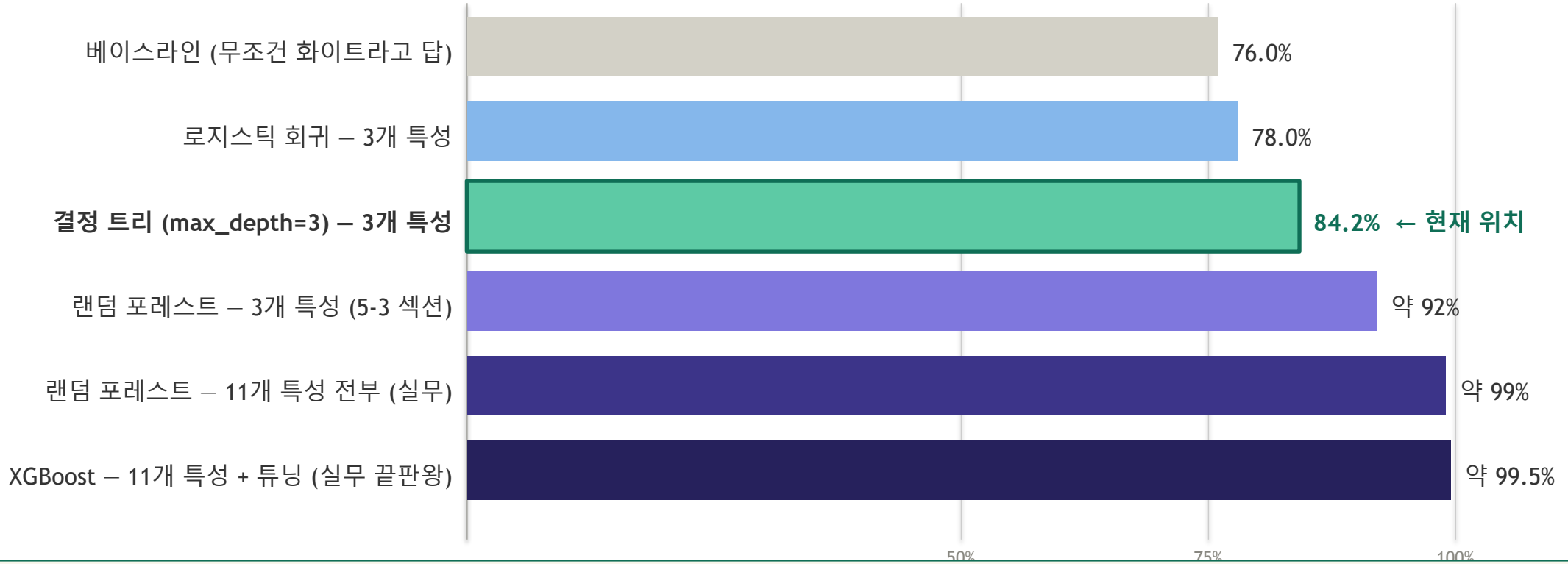
  - 특성 중요도는 각 노드의 정보 이득과 전체 샘플에 대한 비율을 곱한 후 특성별로 더하여 계산
  - 특성 중요도를 활용하면 결정 트리 모델을 특성 선택에 이용할 수 있음

# SECTION 5-1 결정 트리(15)

- 이해하기 쉬운 결정 트리 모델(문제해결 과정)
  - 문제
    - 알코올 도수, 당도, pH 데이터를 기준으로 화이트 와인을 골라내는 이진 분류 로지스틱 회귀 모델을 훈련
    - 설명하고 이해하기 쉬운 방법의 모델이 필요
  - 해결
    - 결정 트리를 사용해 레드 와인과 화이트 와인을 분류하는 문제 해결
    - 특성을 더 추가하지 않고도 결정 트리의 성능이 로지스틱 회귀 모델보다 더 뛰어남
    - 결정 트리는 깊이가 너무 깊지 않다면 비교적 설명하기 쉬움
    - 결정 트리가 어떻게 데이터를 분할하는지 이해하기 위해 불순도 개념과 정보 이득에 대해 학습
    - 결정 트리는 비교적 비전문가에게도 설명하기 쉬운 모델을 만들어줌
    - 결정 트리는 많은 앙상블 학습 알고리즘의 기반이며, 앙상블 학습은 신경망과 함께 가장 높은 성능의 인기 알고리즘

# SECTION 5-1 정확도 사다리 — 우리는 어디쯤 있는가

84%가 끝이 아니라 시작점 — 5-3 섹션과 실무로 가면 99%까지




**베이즈 오차(Bayes error) — 데이터 자체의 한계**

3개 특성만 봤을 때 알코올 11도, 당도 2g/L, pH 3.3인 와인은 가벼운 레드일 수도 드라이한 화이트일 수도 있습니다. 어떤 모델도 100%는 불가능 — 3개 특성 데이터의 이론적 상한은 90~95% 부근. 11개 특성 전부 쓰면 그 상한이 99% 부근으로 올라갑니다.

**결정 트리는 사다리의 시작점 — 빌딩 블록을 잘 이해해야 그 위에 올린 앙상블도 잘 이해됨**

# SECTION 5-1 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - 결정 트리는 예 / 아니오에 대한 질문을 이어나가면서 정답을 찾아 학습하는 알고리즘
    - 비교적 예측 과정을 이해하기 쉽고 성능도 우수
  - 불순도는 결정 트리가 최적의 질문을 찾기 위한 기준
    - 사이킷런은 지니 불순도와 엔트로피 불순도를 제공
  - 정보 이득은 부모 노드와 자식 노드의 불순도 차이
    - 결정 트리 알고리즘은 정보 이득이 최대화되도록 학습
  - 결정 트리는 제한 없이 성장하면 훈련 세트에 과대적합되기 쉬움
    - 가지치기는 결정 트리의 성장을 제한하는 방법
    - 사이킷런의 결정 트리 알고리즘은 여러 가지 가치지기 매개변수를 제공
  - 특성 중요도는 결정 트리에 사용된 특성이 불순도를 감소하는데 기여한 정도를 나타내는 값
    - 특성 중요도를 계산할 수 있는 것이 결정 트리의 또다른 큰 장점

# SECTION 5-1 마무리(2)

- 핵심 패키지와 함수
  - pandas
    - info( ): 데이터프레임의 요약된 정보를 출력
    - describe( ): 데이터프레임 열의 통계 값을 제공
  - scikit-learn
    - DecisionTreeClassifier: 결정 트리 분류 클래스
    - plot_tree( ): 결정 트리 모델을 시각화

# SECTION 5-1 확인 문제

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

   ① 지니 불순도는 부모 노드의 불순도와 자식 노드의 불순도의 차이로 계산
   ② 지니 불순도는 클래스의 비율을 제곱하여 모두 더한 다음1 에서 빼줌
   ③ 엔트로피 불순도는 1에서 가장 큰 클래스 비율을 빼서 계산
   ④ 엔트로피 불순도는 클래스 비율과 클래스 비율에 밑이 2인 로그를 적용한 값을 곱해서 모두 더한 후 음수로 바꾸어 계산

2. 결정 트리에서 계산한 특성 중요도가 저장되어 있는 속성은?

   ① important_variables_
   ② variable_importances_
   ③ important_features_
   ④ feature_importances_

# SECTION 5-1 확인 문제

3. 다음 중 사이킷런의 결정 트리 모델의 최대 깊이를 지정하는 매개변수는 무엇인가요?

① max_depth
② max_leaf_nodes
③ max_features
④ max_samples

# SECTION 5-1 확인 문제

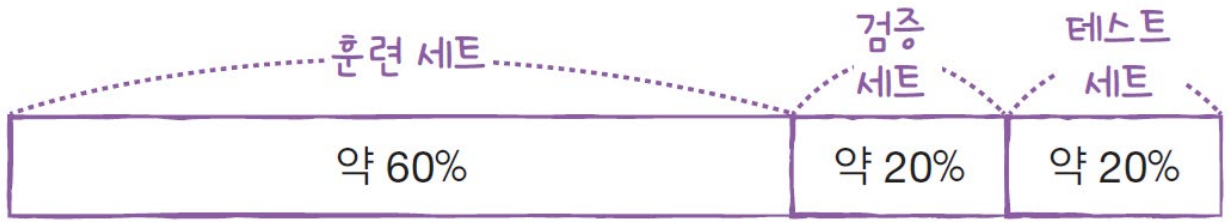
4. 앞서 결정 트리 예제에서 max_depth를 3으로 지정하여 좌우가 대칭인 트리를 만들었습니다. 사이킷런의 결정 트리 클래스가 제공하는 매개변수 중 min_impurity_decrease를 사용해 가지치기를 해 보겠습니다. 어떤 노드의 정보 이득 × (노드의 샘플 수) / (전체 샘플 수) 값이 이 매개변수보다 작으면 더 이상 분할하지 않습니다. 이 매개변수의 값을 0.0005로 지정하고 결정 트리를 만들어 보세요. 좌우가 균일하지 않은 트리가 만들어지나요? 테스트 세트의 성능은 어떤가요?

```
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()
```

# SECTION 5-2 교차 검증과 그리드 서치(1)

◦ 검증 세트

- 테스트 세트를 사용하지 않고 모델이 과대적합인지 과소적합인지 측정하기 위해 다시 나눠진 훈련 세트를 검증 세트(validation set)라고 함
- 앞에서 전체 데이터 중 20%를 테스트 세트로 만들고 나머지 80%를 훈련 세트로 만들었는데, 이 훈련 세트 중에서 다시 20%를 떼어 내어 검증 세트로 만들기
  - 훈련 세트에서 모델을 훈련하고 검증 세트로 모델을 평가



# SECTION 5-2 교차 검증과 그리드 서치(2)

- 검증 세트
  - 검증 세트 만들기
    - 판다스로 CSV 데이터 읽기

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

    - class 열을 타깃으로 사용하고 나머지 열은 특성 배열에 저장

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

  - 훈련 세트와 테스트 세트를 나누기
    - 훈련 세트의 입력 데이터와 타깃 데이터를 train_input과 train_target 배열에 저장

```
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)
```

    - 훈련 세트 sub_input, sub_target과 검증 세트 val_input, val_target을 만들기

```
sub_input, val_input, sub_target, val_target = train_test_split(
train_input, train_target, test_size=0.2, random_state=42)
```

# SECTION 5-2 교차 검증과 그리드 서치(3)

◦ 검증 세트

- 훈련 세트와 테스트 세트를 나누기
  - 훈련 세트와 검증 세트의 크기를 확인

```
print(sub_input.shape, val_input.shape)
```

→ (4157, 3) (1040, 3)

- 모델을 만들고 평가
  - sub_input, sub_target과 val_input, val_target을 사용

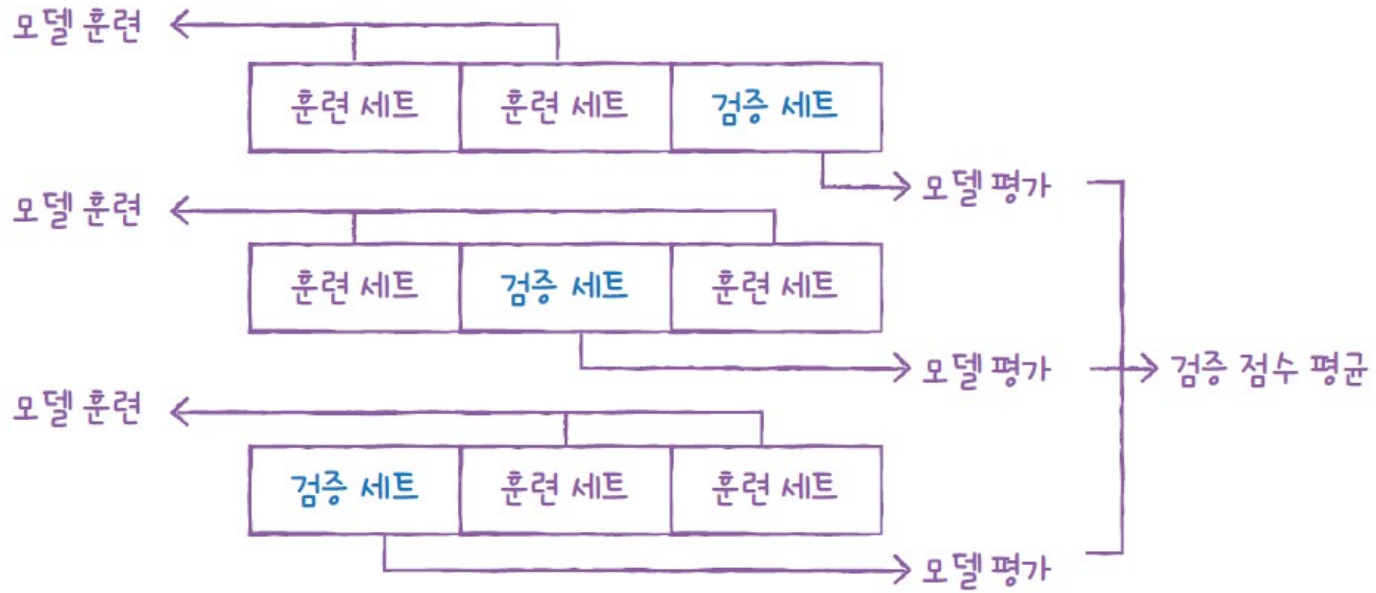
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(sub_input, sub_target)
print(dt.score(sub_input, sub_target))
print(dt.score(val_input, val_target))
```

→
0.9971133028626413
0.864423076923077

# SECTION 5-2 교차 검증과 그리드 서치(4)

◦ 교차 검증(cross validation)

- 교차 검증은 검증 세트를 떼어 내어 평가하는 과정을 여러 번 반복
- 그다음 이 점수를 평균하여 최종 검증 점수를 계산




▲ 3-폴드 교차 검증

# SECTION 5-2 교차 검증 시각화 — 5-폴드의 작동 방식

훈련 세트를 5등분 → 라운드마다 1개를 검증, 4개를 학습 → 5번 평가의 평균

| | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | 검증 점수 |
|---|---|---|---|---|---|---|
| 라운드 1 | 검증 | 학습 | 학습 | 학습 | 학습 | 0.869 |
| 라운드 2 | 학습 | 검증 | 학습 | 학습 | 학습 | 0.846 |
| 라운드 3 | 학습 | 학습 | 검증 | 학습 | 학습 | 0.877 |
| 라운드 4 | 학습 | 학습 | 학습 | 검증 | 학습 | 0.849 |
| 라운드 5 | 학습 | 학습 | 학습 | 학습 | 검증 | 0.836 |

5개 폴드의 검증 점수를 평균 →

**평균 0.855**

**핵심: 5-폴드 = 모델을 5번 학습 + 5번 평가 + 점수 평균**

한 번 검증보다 안정적. 모든 데이터가 학습+검증에 한 번씩 활용됨. 매개변수 한 조합당 5번 학습되므로 그리드 서치 시간 견적의 기준이 됨.

# SECTION 5-2 교차 검증과 그리드 서치(5)

- 교차 검증(cross validation)
  - 사이킷런의 교차 검증 함수 cross_validate( )
    - 평가할 모델 객체를 첫 번째 매개변수로 전달하 다음 앞에서처럼 직접 검증 세트를 떼어 내지 않고 훈련 세트 전체를 cross_validate( ) 함수에 전달

```
from sklearn.model_selection import cross_validate
scores = cross_validate(dt, train_input, train_target)
print(scores)
```

→

```
{' fit_time': array([ 0.01334453,  0.01186419,  0.00783849,  0.0077858,  0.00726461]),
 'score_time': array([0.00085783, 0.00062561,  0.00061512,  0.00063181,  0.00067616]),
 'test_score': array([0.86923077,  0.84615385,  0.87680462,  0.84889317,  0.83541867])}
```

    - 이 함수는 fit_time, score_time, test_score 키를 가진 딕셔너리를 반환
    - 처음 2개의 키는 각각 모델을 훈련하는 시간과 검증하는 시간을 의미
    - 각 키마다 5개의 숫자
    - cross_validate( ) 함수는 기본적으로 5-폴드 교차 검증을 수행(cv 매개변수에서 폴드 수 변경 가능)

# SECTION 5-2 교차 검증과 그리드 서치(6)

◦ 교차 검증(cross validation)

- 사이킷런의 교차 검증 함수 cross_validate( )
  - 교차 검증의 최종 점수는 test_score 키에 담긴 5개의 점수를 평균

```
import numpy as np
print(np.mean(scores['test_score']))
```

→ 0.855300214703487

  - 분할기 지정: cross_validate( ) 함수는 기본적으로 회귀 모델일 경우 KFold 분할기를 사용하고 분류 모델일 경우 타깃 클래스를 골고루 나누기 위해 StratifiedKFold를 사용

```
from sklearn.model_selection import StratifiedKFold
scores = cross_validate(dt, train_input, train_target, cv=StratifiedKFold())
print(np.mean(scores['test_score']))
```

→ 0.855300214703487

  - 훈련 세트를 섞은 후 10-폴드 교차 검증을 수행

n_splits 매개변수는 몇(k) 폴드 교차 검증을 할지 결정

```
splitter = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_validate(dt, train_input, train_target, cv=splitter)
print(np.mean(scores['test_score']))
```

→ 0.8574181117533719

# SECTION 5-2 교차 검증과 그리드 서치(7)

- 하이퍼파라미터 튜닝
  - 모델 파라미터: 머신러닝 모델이 학습하는 파라미터
  - 하이퍼파라미터: 모델이 학습할 수 없어서 사용자가 지정해야만 하는 파라미터
  - 하이퍼파라미터 튜닝 작업의 진행
    - 라이브러리가 제공하는 기본값을 그대로 사용해 모델을 훈련
    - 검증 세트의 점수나 교차 검증을 통해서 매개변수를 조금씩 바꿈
    - 모델마다 적게는 1~2개에서, 많게는 5~6개의 매개변수를 제공
    - 매개변수를 바꿔가면서 모델을 훈련하고 교차 검증을 수행
  - 그리드 서치(Grid Search): max_depth의 최적값은 min_samples_split 매개변수의 값이 바뀌면 함께 달라지므로 이 두 매개변수를 동시에 바꿔가며 최적의 값을 찾기 위해 사용
    - 사이킷런의 GridSearchCV 클래스는 하이퍼파라미터 탐색과 교차 검증을 한 번에 수행

# SECTION 5-2 교차 검증과 그리드 서치(8)

- 하이퍼파라미터 튜닝
  - 기본 매개변수를 사용한 결정 트리 모델에서 min_impurity_decrease 매개변수 최적값 찾기
    - GridSearchCV 클래스를 임포트하고 탐색할 매개변수와 탐색할 값의 리스트를 딕셔너리로 만들기

```
from sklearn.model_selection import GridSearchCV
params = {'min_impurity_decrease': [0.0001, 0.0002, 0.0003, 0.0004, 0.0005]}
```

    - GridSearchCV 클래스에 탐색 대상 모델과 params 변수를 전달하여 그리드 서치 객체 만들기

```
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)
```

    - gs 객체에 fit( ) 메서드를 호출. 이 메서드를 호출하면 그리드 서치 객체는 결정 트리 모델 min_impurity_decrease 값을 바꿔가며 총 5번 실행

```
gs.fit(train_input, train_target)
```

    - 검증 점수가 가장 높은 모델의 매개변수 조합으로 전체 훈련 세트에서 자동으로 다시 모델을 훈련

```
dt = gs.best_estimator_
print(dt.score(train_input, train_target))
```

→ 0.9615162593804117

    - 그리드 서치로 찾은 최적의 매개변수는 best_params_ 속성에 저장됨

```
print(gs.best_params_)
```

→ {'min_impurity_decrease': 0.0001}

# SECTION 5-2 교차 검증과 그리드 서치(9)

- 하이퍼파라미터 튜닝
  - 기본 매개변수를 사용한 결정 트리 모델에서 min_impurity_decrease 매개변수 최적값 찾기
    - 5번의 교차 검증으로 얻은 점수를 출력

```
print(gs.cv_results_['mean_test_score'])
```

→ [0.86819297 0.86453617 0.86492226 0.86780891 0.86761605]

    - 그다음 이 인덱스를 사용해 params 키에 저장된 매개변수를 출력
      (이 값이 최상의 검증 점수를 만든 매개변수 조합)

```
print(gs.cv_results_['params'][gs.best_index_])
```

→ {'min_impurity_decrease': 0.0001}

[과정 요약]

1. 먼저 탐색할 매개변수를 지정
2. 그다음 훈련 세트에서 그리드 서치를 수행하여 최상의 평균 검증 점수가 나오는 매개변수 조합을 찾음(이 조합은 그리드 서치 객체에 저장됨)
3. 그리드 서치는 최상의 매개변수에서 (교차 검증에 사용한 훈련 세트가 아니라) 전체 훈련 세트를 사용해 최종 모델을 훈련(이 모델도 그리드 서치 객체에 저장됨)

# SECTION 5-2 교차 검증과 그리드 서치(10)

- 하이퍼파라미터 튜닝
  - 복잡한 매개변수 조합 탐색
    - min_impurity_decrease는 노드를 분할하기 위한 불순도 감소 최소량을 지정
      max_depth로 트리의 깊이를 제한
      min_samples_split으로 노드를 나누기 위한 최소 샘플 수 선택

```
params = {'min_impurity_decrease': np.arange(0.0001, 0.001, 0.0001),
          'max_depth': range(5, 20, 1),
          'min_samples_split': range(2, 100, 10)
          }
```

① ②

**넘파이 arange( ) 함수(①)**

- 첫 번째 매개변수 값에서 시작하여 두 번째 매개변수에 도달할 때까지 세 번째 매개변수를 계속 더한 배열을 만듦
  코드에서는 0.0001에서 시작하여 0.001이 될 때까지 0.0001을 계속 더한 배열
- 두 번째 매개변수는 포함되지 않으므로 배열의 원소는 총 9개

**파이썬 range( ) 함수(②)**

- 정수만 사용 가능
- 이 경우 max_depth를 5에서 20까지 1씩 증가하면서 15개의 값을 만듦
- min_samples_split은 2에서 100까지 10씩 증가하면서 10개의 값을 만듦
- 따라서 이 매개변수로 수행할 교차 검증 횟수는 9 × 15 × 10 = 1,350개
- 기본 5-폴드 교차 검증을 수행하므로 만들어지는 모델의 수는 모두 6,750개

# SECTION 5-2 교차 검증과 그리드 서치(11)

- 하이퍼파라미터 튜닝
  - 복잡한 매개변수 조합 탐색
    - n_jobs 매개변수를 -1로 설정하고 그리드 서치 실행

```
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)
gs.fit(train_input, train_target)
```

    - 최상의 매개변수 조합 확인

```
print(gs.best_params_)
```

→ {'max_depth': 14, 'min_impurity_decrease': 0.0004, 'min_samples_split': 12}

    - 최상의 교차 검증 점수 확인

```
print(np.max(gs.cv_results_['mean_test_score']))
```

→ 0.8683865773302731

# SECTION 5-2 교차 검증과 그리드 서치(12)

- 하이퍼파라미터 튜닝
  - 랜덤 서치(Random Search)
    - 매개변수의 값이 수치일 때 값의 범위나 간격을 미리 정하기 어려울 때
    - 너무 많은 매개 변수 조건이 있어 그리드 서치 수행 시간이 오래 걸릴 때
    - 랜덤 서치에는 매개변수 값의 목록이 아닌 매개변수를 샘플링할 수 있는 확률 분포 객체를 전달
    - 싸이파이 에서 2개의 확률 분포 클래스를 임포트

```
from scipy.stats import uniform, randint
```

- uniform과 randint 클래스는 모두 주어진 범위에서 고르게 값을 추출함 (균등 분포에서 샘플링)
- randint는 정숫값을 , uniform은 실숫값을 추출

    - 0에서 10 사이의 범위를 갖는 randint 객체를 만들고 10개의 숫자를 샘플링

```
rgen = randint(0, 10)
rgen.rvs(10)
```

→ array([6, 4, 2, 2, 7, 7, 0, 0, 5, 4])

    - 1,000개를 샘플링해서 각 숫자의 개수 구하기

```
np.unique(rgen.rvs(1000),
return_counts=True)
```

→ (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([ 98, 94, 99, 93, 93, 92, 111, 118, 105, 97]))

# SECTION 5-2 교차 검증과 그리드 서치(13)

- 하이퍼파라미터 튜닝
  - 랜덤 서치(Random Search)
    - uniform 클래스로 0~1 사이에서 10개의 실수를 추출

```
ugen = uniform(0, 1)
ugen.rvs(10)
```

→

```
array([0.12982148, 0.32130647, 0.22468098, 0.09345374, 0.43188927,
       0.69791727, 0.81250121, 0.54913255, 0.00552007, 0.52386115])
```

    - 탐색할 매개변수의 딕셔너리를 만들기
      - min_samples_leaf 매개변수를 탐색 대상에 추가

```
params = {'min_impurity_decrease': uniform(0.0001, 0.001),
'max_depth': randint(20, 50),
'min_samples_split': randint(2, 25),
'min_samples_leaf': randint(1, 25),
}
```

      - 사이킷런의 랜덤 서치 클래스 RandomizedSearchCV의 n_iter 매개변수에 샘플링 횟수 지정

```
from sklearn.model_selection import RandomizedSearchCV
rs = RandomizedSearchCV(DecisionTreeClassifier(random_state=42), params,
n_iter=100, n_jobs=-1, random_state=42)
rs.fit(train_input, train_target)
```

# SECTION 5-2 교차 검증과 그리드 서치(14)

◦ 하이퍼파라미터 튜닝

- 랜덤 서치(Random Search)

  • params에 정의된 매개변수 범위에서 총 100번(n_iter 매개변수)을 샘플링하여 교차 검증을 수행하고 최적의 매개변수 조합 탐색

```
print(rs.best_params_)
```

→ { 'max_depth': 39, 'min_impurity_decrease': 0.00034102546602601173, 'min_samples_leaf': 7, 'min_samples_split': 13}

  • 최고의 교차 검증 점수 확인

```
print(np.max(rs.cv_results_['mean_test_score']))
```

→ 0.8695428296438884

  • 최종 모델로 결정하고 테스트 세트의 성능 확인

```
dt = rs.best_estimator_
print(dt.score(test_input, test_target))
```

→ 0.86

# SECTION 5-2 교차 검증과 그리드 서치(15)

- 최적의 모델을 위한 하이퍼파라미터 탐색(문제해결 과정)
  - 문제
    - 레드 와인과 화이트 와인 선별 작업의 성능 향상을 위해 결정 트리의 다양한 하이퍼파라미터를 시도
  - 해결
    - 테스트 세트를 사용하지 않고 모델을 평가하기 위해 검증 세트(혹은 개발 세트, dev set) 이용
    - 교차 검증: 훈련한 모델의 성능을 안정적으로 평가하기 위해 검증 세트를 한 번 나누어 모델을 평가하는 것에 그치지 않고 여러 번 반복
      - 보통 훈련 세트를 5등분 혹은 10등분. 전체적으로 5개 혹은 10개의 모델을 만들고 최종 검증 점수는 모든 폴드의 검증 점수를 평균하여 계산
    - 교차 검증을 사용해 다양한 하이퍼파라미터를 탐색
      - 클래스와 메서드의 매개변수를 바꾸어 모델을 훈련하고 평가
    - 그리드 서치를 사용하여 테스트하고 싶은 매개변수 리스트를 만들어 이 과정을 자동화
    - 매개변수 값이 수치형이고 특히 연속적인 실숫값이라면 싸이파이의 확률 분포 객체를 전달하여 특정 범위 내에서 지정된 횟수만큼 매개변수 후보 값을 샘플링하여 교차 검증을 시도

# SECTION 5-2 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - 검증 세트는 하이퍼파라미터 튜닝을 위해 모델을 평가할 때, 테스트 세트를 사용하지 않기 위해 훈련 세트에서 다시 떼어 낸 데이터 세트
  - 교차 검증은 훈련 세트를 여러 폴드로 나눈 다음 한 폴드가 검증 세트의 역할을 하고 나머지 폴드에서는 모델을 훈련
    - 교차 검증은 이런 식으로 모든 폴드에 대해 검증 점수를 얻어 평균하는 방법
  - 그리드 서치는 하이퍼파라미터 탐색을 자동화해 주는 도구
    - 탐색할 매개변수를 나열하면 교차 검증을 수행하여 가장 좋은 검증 점수의 매개변수 조합을 선택
    - 마지막으로 이 매개변수 조합으로 최종 모델을 훈련
  - 랜덤 서치는 연속된 매개변수 값을 탐색할 때 유용
    - 탐색할 값을 직접 나열하는 것이 아니고 탐색 값을 샘플링할 수 있는 확률 분포 객체를 전달
    - 지정된 횟수만큼 샘플링하여 교차 검증을 수행하기 때문에 시스템 자원이 허락하는 만큼 탐색량 조절 가능

# SECTION 5-2 마무리(2)

- 핵심 패키지와 함수
  - scikit-learn
    - cross_validate( ): 교차 검증을 수행하는 함수
    - GridSearchCV: 교차 검증으로 하이퍼파라미터 탐색을 수행
    - RandomizedSearchCV: 교차 검증으로 랜덤한 하이퍼파라미터 탐색을 수행

# SECTION 5-2 확인 문제

1. 훈련 세트를 여러 개의 폴드로 나누고 폴드 1개는 평가 용도로, 나머지 폴드는 훈련 용도로 사용하고, 그다음 모든 폴드를 평가 용도로 사용하게끔 폴드 개수만큼 이 과정을 반복.
   이런 평가 방법을 무엇이라고 부르나?

   ① 교차 검증　　② 반복 검증
   ③ 교차 평가　　④ 반복 평가

2. 다음 중 교차 검증을 수행하지 않는 함수나 클래스는 무엇인가?

   ① cross_validate( )
   ② GridSearchCV
   ③ RandomizedSearchCV
   ④ train_test_split

# SECTION 5-2 확인 문제

3. 다음 중 GridSearchCV에 대해 올바르게 설명한 것은 무엇인가요?

1) 사람의 개입없이 하이퍼파라미터를 자동으로 검색합니다.

2) 확률 분포에서 후보 매개변수를 샘플링합니다.

3) 기본적으로 5-폴드 교차 검증을 수행하여 최적의 하이퍼파라미터 조합을 찾습니다.

4) DecisionTreeClassifier만 사용할 수 있습니다.

4. 마지막 RandomizedSearchCV 예제에서 DecisionTreeClassifier 클래스에 splitter='random' 매개변수를 추가하고 다시 훈련해 보세요. splitter 매개변수의 기본값은 'best'로 각 노드에서 최선의 분할을 찾습니다. 'random'이면 무작위로 분할한 다음 가장 좋은 것을 고릅니다. 왜 이런 옵션이 필요한지는 다음 절에서 알 수 있습니다. 테스트 세트에서 성능이 올라갔나요? 내려갔나요?

# SECTION 5-3 트리의 앙상블(1)

- 정형 데이터와 비정형 데이터
  - 정형 데이터(structured data)
    - 어떤 구조로 되어 있는 생선의 길이, 높이, 무게, 와인 데이터 등
    - CSV나 데이터베이스(Database), 혹은 엑셀(Excel)에 저장하기 용이함
    - 머신러닝 알고리즘과 앙상블 학습(ensemble learning)
    - 이 알고리즘은 대부분 결정 트리를 기반으로 만들어짐
  - 비정형 데이터(unstructured data)
    - 텍스트 데이터, 디지털카메라로 찍은 사진, 핸드폰으로 듣는 디지털 음악 등
    - 신경망 알고리즘

# SECTION 5-3 앙상블의 직관 — 다수결의 힘

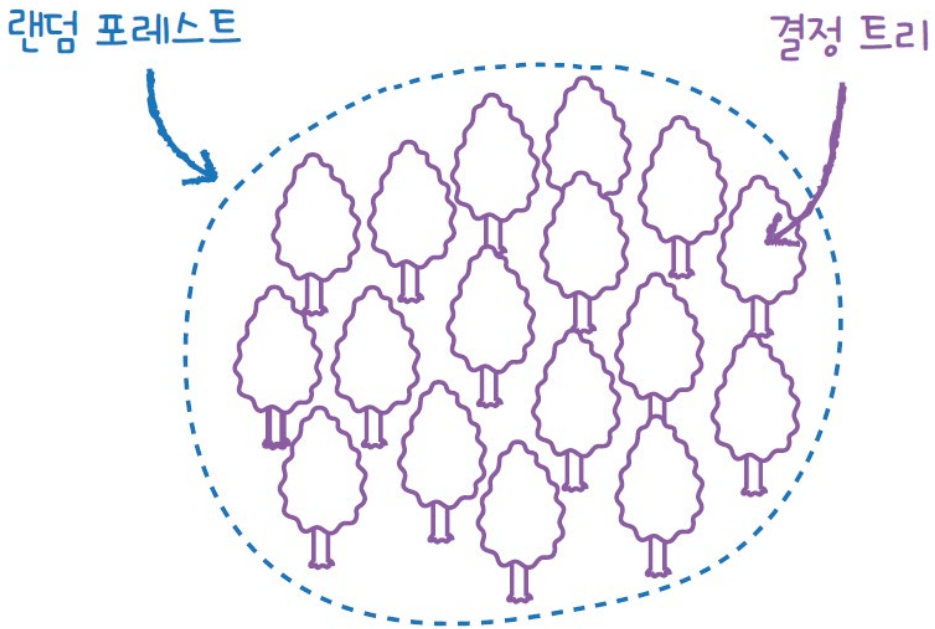
약한 모델 여러 개의 다수결이 강한 모델 한 개보다 정확하다


**Condorcet 배심원 정리 (1785): 51% 정확한 분류기를 N개 모아 다수결하면, N → ∞일 때 정확도 → 100%. 단, 오차가 서로 독립적일 때만 성립 — 그래서 "다양성"이 핵심.**

# SECTION 5-3 트리의 앙상블(2)

◦ 랜덤 포레스트(Random Forest)

- 대표적인 앙상블 학습
  - 안정적인 성능 덕분에 널리 사용됨
  - 랜덤 포레스트는 결정 트리를 랜덤하게 만들어 결정 트리(나무)의 숲을 만들고 각 결정 트리의 예측을 사용해 최종 예측을 도출



# SECTION 5-3 트리의 앙상블(3)

- 랜덤 포레스트(Random Forest)
  - 부트스트랩 샘플(bootstrap sample)
    - 데이터 세트에서 중복을 허용하여 데이터를 샘플링하는 방식
    - 기본적으로 부트스트랩 샘플은 훈련세트의 크기와 같게 만듦
    - 1,000개 가방에서 중복하여 1,000개의 샘플을 뽑기 때문에 부트스트랩 샘플은 훈련 세트와 크기가 같음
    - 각 노드를 분할할 때 전체 특성 중에서 일부 특성을 무작위로 고른 다음 이 중에서 최선의 분할을 탐색
  - 분류 모델인 RandomForestClassifier는 기본적으로 전체 특성 개수의 제곱근만큼의 특성을 선택
    - 즉 4개의 특성이 있다면 노드마다 2개를 랜덤하게 선택하여 사용
  - 회귀 모델인 RandomForestRegressor는 전체 특성을 사용
  - 사이킷런의 랜덤 포레스트는 기본적으로 100개의 결정 트리를 이런 방식으로 훈련
    - 분류일 때는 각 트리의 클래스별 확률을 평균하여 가장 높은 확률을 가진 클래스를 예측으로
    - 회귀일 때는 단순히 각 트리의 예측을 평균

# SECTION 5-3 트리의 앙상블(4)

◦ 랜덤 포레스트(Random Forest)

- RandomForestClassifier 클래스를 화이트 와인을 분류하는 문제에 적용하기
  - 와인 데이터셋을 판다스로 불러오고 훈련 세트와 테스트 세트로 나누기

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
wine = pd.read_csv('https://bit.ly/wine-date')
data = wine[['alcohol', 'sugar', 'pH']]
target = wine['class']
train_input, test_input, train_target, test_target = train_test_split(
    data, target, test_size=0.2, random_state=42)
```

  - cross_validate( ) 함수를 사용해 교차 검증을 수행

```
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs=-1, random_state=42)
scores = cross_validate(rf, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

→ 0.9973541965122431 0.8905151032797809

# SECTION 5-3 트리의 앙상블(5)

◦ 랜덤 포레스트(Random Forest)

▪ RandomForestClassifier 클래스를 화이트 와인을 분류하는 문제에 적용하기

• 앞의 랜덤 포레스트 모델을 훈련 세트에 훈련한 후 특성 중요도를 출력

```
rf.fit(train_input, train_target)
print(rf.feature_importances_)
```

→ [0.23167441 0.50039841 0.26792718]

[0.12345626 0.86862934 0.00791441]

앞의 1절 '결정 트리'에서 만든 특성 중요도(본문 234쪽)

▲ 각각 [알코올 도수, 당도, pH], 두 번째 특성인 당도의 중요도가 감소하고 알코올 도수와 pH 특성의 중요도가 조금 상승

이런 이유는 랜덤 포레스트가 특성의 일부를 랜덤하게 선택하여 결정 트리를 훈련하기 때문임

그 결과 하나의 특성에 과도하게 집중하지 않고 좀 더 많은 특성이 훈련에 기여할 기회를 획득하므로 과대적합을 줄이고 일반화 성능을 높이는 데 도움

# SECTION 5-3 트리의 앙상블(6)

◦ 랜덤 포레스트(Random Forest)

- RandomForestClassifier 클래스를 화이트 와인을 분류하는 문제에 적용하기
  - OOB(out of bag) 샘플: 부트스트랩 샘플에 포함되지 않고 남는 샘플
  - 부트스트랩 샘플로 훈련한 결정 트리를 평가하는데 사용
  - oob_score=True로 지정하고 모델을 훈련하여 OOB 점수 출력

```
rf = RandomForestClassifier(oob_score=True, n_jobs=-1,
random_state=42)
rf.fit(train_input, train_target)
print(rf.oob_score_)
```

→ 0.8934000384837406

# SECTION 5-3 트리의 앙상블(7)

- 엑스트라 트리(Extra Trees)
  - 기본적으로 100개의 결정 트리를 훈련하고, 랜덤 포레스트와 동일하게 결정 트리가 제공하는 대부분의 매개변수를 지원
  - 전체 특성 중에 일부 특성을 랜덤하게 선택하여 노드를 분할하는 데 사용
  - 랜덤 포레스트와 엑스트라 트리의 차이점은 부트스트랩 샘플을 사용하지 않는다는 점
  - ExtraTreesClassifier를 사용해 모델의 교차 검증 점수를 확인

```
from sklearn.ensemble import ExtraTreesClassifier
et = ExtraTreesClassifier(n_jobs=-1, random_state=42)
scores = cross_validate(et, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

→ 0.9974503966084433 0.8887848893166506

  - 엑스트라 트리도 랜덤 포레스트와 마찬가지로 특성 중요도를 제공. 순서는 [알코올 도수, 당도, pH]인데, 결과를 보면 엑스트라 트리도 결정 트리보다 당도에 대한 의존성이 작음

```
et.fit(train_input, train_target)
print(et.feature_importances_)
```

→ [0.20183568 0.52242907 0.27573525]

# SECTION 5-3 트리의 앙상블(8)

- 그레이디언트 부스팅(gradient boosting)
  - 깊이가 얕은 결정 트리를 사용하여 이전 트리의 오차를 보완하는 방식으로 앙상블 하는 방법
  - 경사 하강법을 사용하여 트리를 앙상블에 추가
  - 분류에서는 로지스틱 손실 함수를 사용하고 회귀에서는 평균 제곱 오차 함수를 사용
  - GradientBoostingClassifier를 사용해 와인 데이터셋의 교차 검증 점수 확인

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(random_state=42)
scores = cross_validate(gb, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']),
np.mean(scores['test_score']))
```

→ 0.8881086892152563
0.8720430147331015

  - 그레이디언트 부스팅은 결정 트리의 개수를 늘려도 과대적합에 매우 강함
  - 결정 트리 개수를 500개로 늘려서 교차 검증 점수 확인

```
gb = GradientBoostingClassifier(n_estimators=500, learning_rate=0.2,
                                random_state=42)
scores = cross_validate(gb, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

→ 0.9464595437171814 0.8780082549788999

# SECTION 5-3 트리의 앙상블(9)

◦ 그레이디언트 부스팅(gradient boosting)

- 그레이디언트 부스팅도 특성 중요도 확인
  (결과에서 볼 수 있듯이 그레이디언트 부스팅이 랜덤 포레스트보다 일부 특성(당도)에 더 집중)

```
gb.fit(train_input, train_target)
print(gb.feature_importances_)
```

→ [0.15887763 0.6799705  0.16115187]

- subsample: 트리 훈련에 사용할 훈련 세트의 비율을 정하는 매개변수
  - 이 매개변수의 기본값은 1.0으로 전체 훈련 세트를 사용
  - subsample이 1보다 작으면 훈련 세트의 일부를 사용

# SECTION 5-3 트리의 앙상블(10)

- 히스토그램 기반 그레이디언트 부스팅(Histogram-based Gradient Boosting)
  - 정형 데이터를 다루는 머신러닝 알고리즘 중에 가장 높은 인기
  - 입력 특성을 256개의 구간으로 나누어 노드를 분할할 때 최적의 분할을 매우 빠르게 탐색
  - 256개의 구간 중에서 하나를 떼어 놓고 누락된 값을 위해서 사용
  - 따라서 입력에 누락된 특성이 있더라도 이를 따로 전처리할 필요가 없음
  - 와인 데이터셋에 HistGradientBoostingClassifier 클래스를 적용

```
from sklearn.ensemble import HistGradientBoostingClassifier
hgb = HistGradientBoostingClassifier(random_state=42)
scores = cross_validate(hgb, train_input, train_target,
                        return_train_score=True)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

→ 0.9321723946453317 0.880124194861923

# SECTION 5-3 트리의 앙상블(11)

◦ 히스토그램 기반 그레이디언트 부스팅(Histogram-based Gradient Boosting)

- 히스토그램 기반 그레이디언트 부스팅 모델을 훈련하고 훈련 세트에서 특성 중요도를 계산
  n_repeats 매개변수는 랜덤하게 섞을 횟수를 지정

```
from sklearn.inspection import permutation_importance
hgb.fit(train_input, train_target)
result = permutation_importance(hgb, train_input, train_target,
                                n_repeats=10, random_state=42, n_jobs=-1)
print(result.importances_mean)
```

→ [0.08876275 0.23438522
0.08027708]

- 테스트 세트에서 특성 중요도를 계산

```
result = permutation_importance(hgb, test_input, test_target, n_repeats=10,
                                random_state=42, n_jobs=-1)
print(result.importances_mean)
```

→ 0.8723076923076923

- 테스트 세트의 결과를 보면 그레이디언트 부스팅과 비슷하게 조금 더 당도에 집중하고 있다는 것을 알 수 있음. 이런 분석을 통해 모델을 실전에 투입했을 때 어떤 특성에 관심을 둘지 예상할 수 있음

# SECTION 5-3 트리의 앙상블(12)

◦ 히스토그램 기반 그레이디언트 부스팅(Histogram-based Gradient Boosting)

- HistGradientBoostingClassifier를 사용해 테스트 세트에서의 성능을 최종적으로 확인

```
hgb.score(test_input, test_target)
```

→ 0.8723076923076923

▪ 다른 라이브러리

- XGBoost 라이브러리를 사용해 와인 데이터의 교차 검증 점수 확인

```
from xgboost import XGBClassifier
xgb = XGBClassifier(tree_method='hist', random_state=42)
scores = cross_validate(xgb, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

→ 0.9558403027491312 0.8782000074035686

- 마이크로소프트 LightGBM 라이브러리를 사용해 와인 데이터의 교차 검증 점수 확인

```
from lightgbm import LGBMClassifier
lgb = LGBMClassifier(random_state=42)
scores = cross_validate(lgb, train_input, train_target,
                        return_train_score=True, n_jobs=-1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
```

→ 0.935828414851749 0.8801251203079884

# SECTION 5-3 트리의 앙상블(13)

- 앙상블 학습을 통한 성능 향상(문제해결 방식)
  - 앙상블 학습은 정형 데이터에서 가장 뛰어난 성능을 내는 머신러닝 알고리즘 중 하나
  - 사이킷런
    - 랜덤 포레스트: 부트스트랩 샘플 사용. 대표 앙상블 학습 알고리즘
    - 엑스트라 트리: 결정 트리의 노드를 랜덤하게 분할함
    - 그레이디언트 부스팅: 이진 트리의 손실을 보완하는 식으로 얕은 결정 트리를 연속하여 추가
    - 히스토그램 기반 그레이디언트 부스팅: 훈련 데이터를 256개 정수 구간으로 나누어 빠르고 높은 성능
  - 그외 라이브러리
    - XGBoost
    - LightGBM

# SECTION 5-3 앙상블 알고리즘 한눈에 비교

배깅 vs 부스팅 – 6가지 알고리즘의 핵심 특징을 한 표에

| 알고리즘 | 카테고리 | 부트스트랩 | 학습 속도 | 튜닝 민감도 | 결측값 자동 |
|---|---|---|---|---|---|
| 랜덤 포레스트 | 배깅 | ✓ | 빠름 | 낮음 (안전) | ✗ |
| 엑스트라 트리 | 배깅 | ✗ | 매우 빠름 | 낮음 | ✗ |
| 그레이디언트 부스팅 | 부스팅 | ✗ | 느림 | 높음 | ✗ |
| 히스토그램 GB | 부스팅 | ✗ | 매우 빠름 | 중간 | ✓ |
| XGBoost | 부스팅 | ✗ | 매우 빠름 | 높음 | ✓ |
| LightGBM | 부스팅 | ✗ | 극도로 빠름 | 중간 | ✓ |

**배깅 (Bagging) 계열**

트리들을 독립적·병렬로 만들어 다수결/평균. 매개변수 튜닝 부담이 작아 베이스라인으로 적합. 분산 감소가 강점.

**부스팅 (Boosting) 계열**

트리들을 순차적으로 만들어 이전 모델의 오차를 보완. 튜닝 시 더 높은 성능. 캐글 정형 데이터의 표준.

# SECTION 5-3 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - 앙상블 학습은 더 좋은 예측 결과를 만들기 위해 여러 개의 모델을 훈련하는 머신러닝 알고리즘
  - 랜덤 포레스트는 대표적인 결정 트리 기반의 앙상블 학습 방법
    - 부트스트랩 샘플을 사용하고 랜덤하게 일부 특성을 선택하여 트리를 만드는 것이 특징
  - 엑스트라 트리는 랜덤 포레스트와 비슷하게 결정 트리를 사용하여 앙상블 모델을 만들지만 부트스트랩 샘플을 사용하지 않음
    - 대신 랜덤하게 노드를 분할해 과대적합을 감소
  - 그레이디언트 부스팅은 랜덤 포레스트나 엑스트라 트리와 달리 결정 트리를 연속적으로 추가하여 손실 함수를 최소화하는 앙상블 방법
    - 이런 이유로 훈련 속도가 조금 느리지만 더 좋은 성능을 기대할 수 있음
    - 그레이디언트 부스팅의 속도를 개선한 것이 히스토그램 기반 그레이디언트 부스팅이며 안정적인 결과와 높은 성능으로 매우 인기가 높음

# SECTION 5-3 마무리(2)

- 핵심 패키지와 함수
  - scikit-learn
    - RandomForestClassifier: 랜덤 포레스트 분류 클래스
    - ExtraTreesClassifier: 엑스트라 트리 분류 클래스
    - GradientBoostingClassifier: 그레이디언트 부스팅 분류 클래스
    - HistGradientBoostingClassifier: 히스토그램 기반 그레이디언트 부스팅 분류 클래스

# SECTION 5-3 확인 문제

1. 여러 개의 모델을 훈련시키고 각 모델의 예측을 취합하여 최종 결과를 만드는 학습 방식을 무엇이라고 부르나요?

① 단체 학습
② 오케스트라 학습
③ 심포니 학습
④ 앙상블 학습

1. 다음 중 비정형 데이터에 속하는 것은?

① 엑셀 데이터
② CSV 데이터
③ 데이터베이스 데이터
④ 이미지 데이터

# SECTION 5-3 확인 문제

3. 다음 알고리즘 중 기본적으로 부트스트랩 샘플을 사용하는 알고리즘은?

① 랜덤 포레스트
② 엑스트라 트리
③ 그레이디언트 부스팅
④ 히스토그램 기반 그레이디언트 부스팅

4. 다음 중 순서대로 트리를 추가하여 앙상블 모델을 만드는 방법은 무엇인가요?

① 결정 트리
② 랜덤 포레스트
③ 엑스트라 트리
④ 그레이디언트 부스팅