트리의 앙상블


구글 코랩에서 실행하기

랜덤포레스트

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

wine = pd.read_csv('https://bit.ly/wine_csv_data')

data = wine[['alcohol', 'sugar', 'pH']].to_numpy() # 알코올, 당도, PH를 데이터에 저장
target = wine['class'].to_numpy() # class 열을 target에 저장

train_input, test_input, train_target, test_target = train_test_split(data, target, test_size=0.2, random_state=42) # 훈련, 테스트 세트를 8:2로 나눔
```

In [3]:
```
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_jobs=-1, random_state=42) # n_jobs=-1로 하여 모든 CPU코어 사용 및 최대한 병렬로 교차 검증 수행
scores = cross_validate(rf, train_input, train_target, return_train_score=True, n_jobs=-1) # return_train_score=True를 통해 검증 점수 뿐만 아니라 훈련 세트에 대한 점수도 같이 반환

print(np.mean(scores['train_score']), np.mean(scores['test_score'])) # overfitting
```

```
0.9973541965122431 0.8905151032797809
```

In [4]:
```
rf.fit(train_input, train_target) # rf를 훈련
print(rf.feature_importances_) # 특성 중요도 출력
# 결정트리의 중요도와 다른 결과
# -> 랜덤 포레스트가 특성의 일부를 랜덤하게 선택하여 결정트리를 훈련하기 때문
# --> 하나의 특성에 과도하게 집중하지 않고 좀 더 많은 특성 훈련에 기여할 기회
# --> 과대 적합을 줄이고 성능을 높이는데 도움
```

```
[0.23167441 0.50039841 0.26792718]
```

In [5]:
```
rf = RandomForestClassifier(oob_score=True, n_jobs=-1, random_state=42)
# oob_score = True를 통해 부트스트랩 샘플에 포함되지 않고 남은 샘플로 결정 트리를 평가
# oob 점수를 사용하면 교차 검증을 대신 할 수 있어서 결과적으로 훈련 세트에 더 많은 샘플을 사용가능
rf.fit(train_input, train_target)
print(rf.oob_score_)
```

```
0.8934000384837406
```

엑스트라트리

- 엑스트라 트리는 랜덤 포레스트와 비슷하게 동작
- 단, 부트스트랩 샘플을 사용하지 않고 각 결정트리를 만들 때 전체 훈련 세트를 사용
- 대신, 노드를 분할할 때 가장 좋은 분할을 찾는 것이 x, 무작위로 분할

In [6]:
```
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
```

In [7]:
```
et.fit(train_input, train_target) # 엑스트라 트리는 무작위성이 좀 더 크기 때문에 랜덤 포레스트보다 더 많은 결정 트리를 훈련해야 함
print(et.feature_importances_) # 하지만, 랜덤하게 노드를 분할하기 때문에 빠른 계산속도가 엑스트라 트리의 장점
```

```
[0.20183568 0.52242907 0.27573525]
```

그레이디언트 부스팅

- 그래디언트 부스팅은 깊이가 얕은 결정 트리를 사용하여 이전 트리의 잔차를 보완하는 방식으로 앙상블 하는 방법
- GradientBoostingClassifier는 기본적으로 깊이가 3인 결정 트리를 100개 사용
  - 과대 적합에 강하고, 높은 일반화 성능을 기대 가능
- 경사 하강법으로 트리를 앙상블에 추가
  - 분류에서는 로지스틱 손실 함수 사용
  - 회귀에서는 평균 제곱 오차 함수 사용

In [8]:
```
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
```

In [9]:
```
gb = GradientBoostingClassifier(n_estimators=500, learning_rate=0.2, random_state=42) # n_estimators=500으로 하여 결정 트리 개수를 500으로 5배 늘림
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
```

In [10]:
```
gb.fit(train_input, train_target)
print(gb.feature_importances_)
# 그래디언트 부스팅이 랜덤 포레스트보다 일부 특성(당도)에 더 집중
# 랜덤포레스트보다 조금 더 높은 성능, 하지만 순서대로 트리를 추가하기 때문에 훈련 속도가 느림
# subsample이라는 매개변수도 존재
# -> 기본값은 1.0으로 전체 훈련 세트를 사용하지만, 1보다 작으면 훈련 세트의 일부를 사용
```

```
[0.15872278 0.68010884 0.16116839]
```

히스토그램 기반 부스팅

XGBoost

In [11]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier(tree_method='hist', random_state=42)
# tree_method(트리 빌딩방법) : split point를 잡을 때 히스토그램의 사용 여부
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
```

LightGBM

In [12]:
```
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
```

In [ ]:
```

```