week4. 트리 알고리즘

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

	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

▼ pandas 의 핵심 패키지와 함수

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

info () 는 데이터프레임의 요약된 정보를 출력. 인덱스와 컬럼 타입을 출력하고 널(null)이 아닌 값의 개수, 메모리 사용량을 제공.

verbose 매개변수의 기본값 True 를 False 로 바꾸면 각 열에 대한 정보를 출력하지 않음

describe() 메서드 : 열에 대한 간략한 통계 출력 , 최소, 최대, 평균값 확인

describe() 는 데이터프레임 열의 통계 값을 제공. 수치형일 경우 최소, 최대, 평균, 표준편차와 사분위값등이 출력됨.

문자열 같은 객체 타입의 열은 가장 자주 등장하는 값과 횟수 등이 출력됨.

percentiles 매개변수에서 백분위수를 지정. 기본값은 [0.25, 0.5, 0.75]

▼ 결정 트리

예/아니오 에 대한 질문을 이어나가면서 정답을 찾아 학습하는 알고리즘.

비교적 예측 과정을 이해하기 쉽고 성능이 뛰어남.

결정 트리의 장점 : 표준화 전처리를 할 필요가 없다

특성값의 스케일은 결정 트리 알고리즘에 아무런 영향을 미치지 않는다.

불순도 :

결정 트리가 최적의 질문을 찾기 위한 기준.

사이킷런은 지니 불순도와 엔트로피 불순도를 제공.

$$\text{지니불순도} = 1 - (\text{음성클래스 비율}^2 + \text{양성클래스 비율}^2)$$

$$1 - \left(\left(\frac{1258}{5197}\right)^2 + \left(\frac{3939}{5197}\right)^2\right) = 0.367$$

$$1 - \left(\left(\frac{50}{100}\right)^2 + \left(\frac{50}{100}\right)^2\right) = 0.5$$

$$1 - \left(\left(\frac{0}{100}\right)^2 + \left(\frac{100}{100}\right)^2\right) = 0$$

$$\text{부모의 불순도} - \frac{\text{왼쪽 노드의 샘플수}}{\text{부모의 샘플수}} \times \text{왼쪽 노드의 불순도} - \frac{\text{오른쪽 노드의 샘플수}}{\text{부모의 샘플수}} \times \text{오른쪽 노드의 불순도}$$

→ 지니 불순도가 0일 때 , 순수노드

정보 이득 :

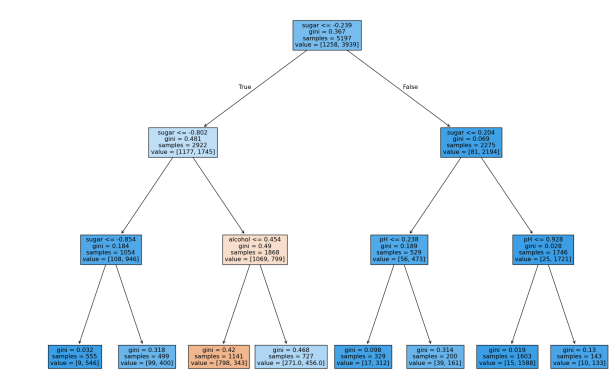
부모 노드와 자식 노드의 불순도 차이.

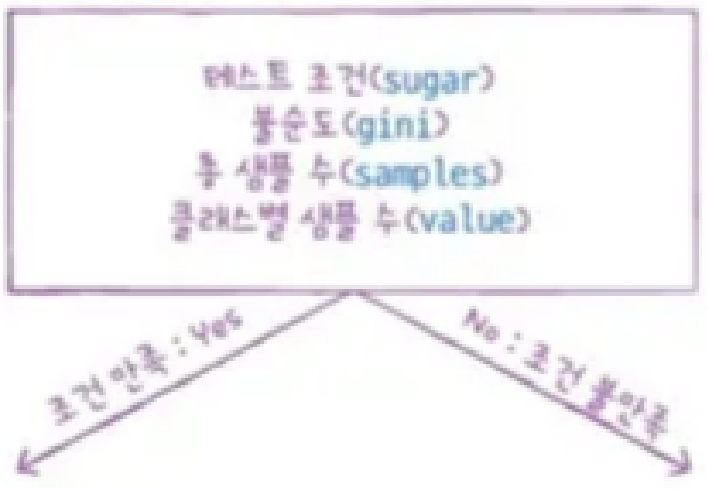
결정 트리 알고리즘은 정보 이득이 최대화되도록 학습.

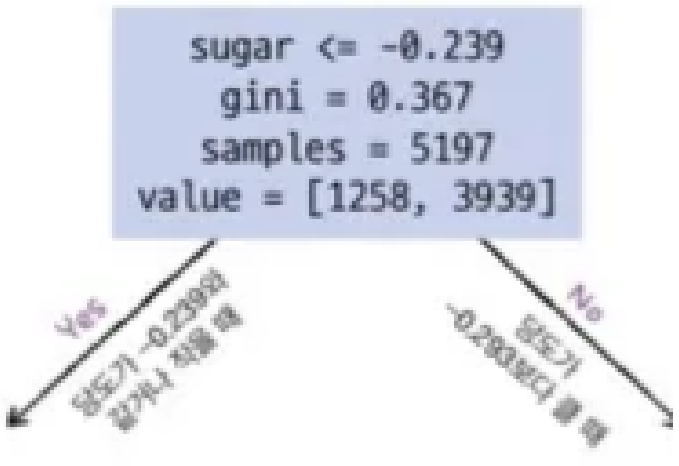
결정 트리는 제한 없이 성장하면 훈련 세트에 과대적합 되기 쉬움.

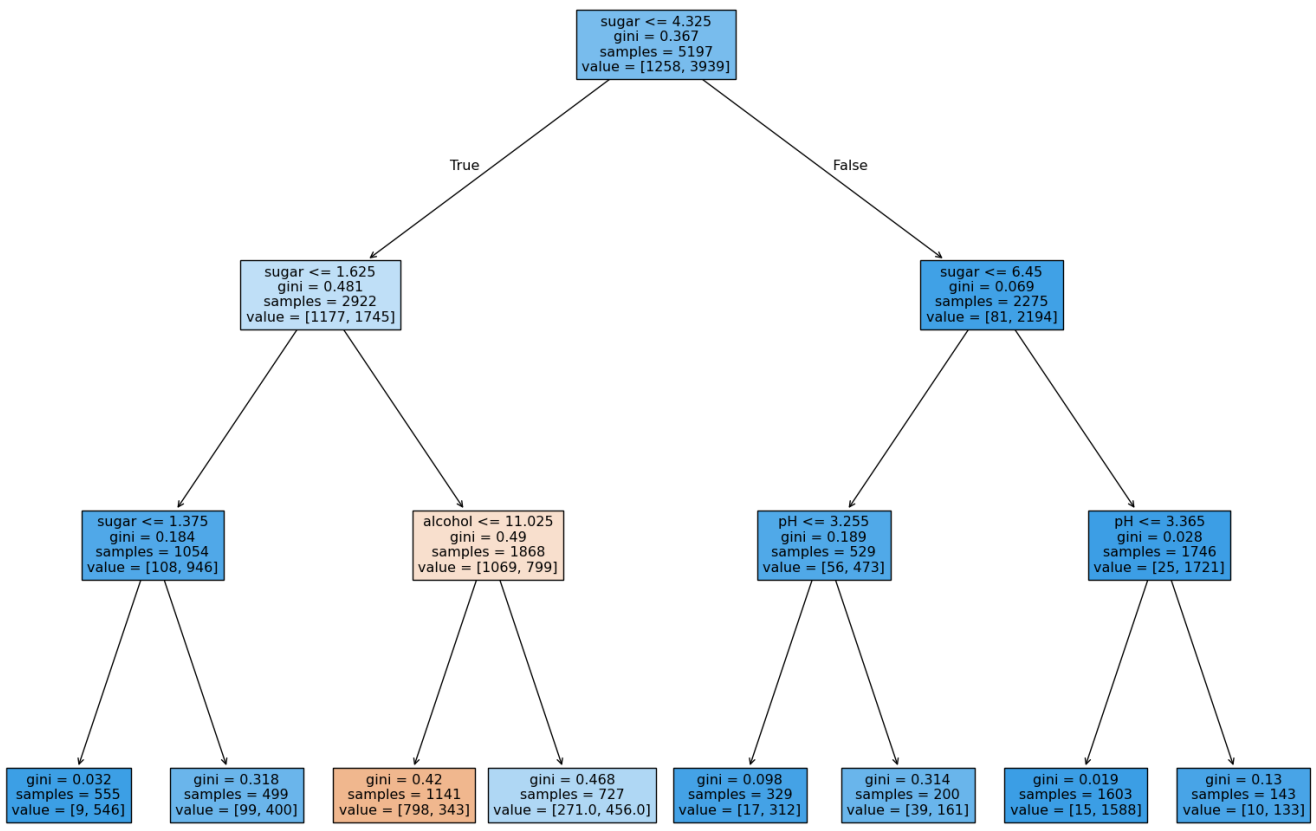
가지치기 :

결정 트리의 성장을 제한하는 방법. 사이킷런의 결정 트리 알고리즘은 여러 가지 가지치기 매개변수를 제공.









사이킷런이 제공하는 결정 트리 알고리즘 : **사이킷런의 DecisionTreeClassifier 클래스**

```
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
```

사이킷런의 plot_tree() 함수

→ 결정 트리를 이해하기 쉬운 트리 그림으로 출력

max_depth 매개변수 : 트리의 깊이 지정 , 기본값 : None

feature_names 매개변수 : 특성 이름 지정

filled 매개변수를 True 로 지정 → 타깃값에 따라 노드 안에 색 채움

특성 중요도 :

결정 트리에 사용된 특성이 불순도를 감소하는데 기여한 정도.

특성 중요도를 계산 할 수 있는 것이 결정 트리의 장점.

▼ 검증 세트

하이퍼파라미터 튜닝을 위해 모델을 평가할 때, 테스트 세트를 사용하지 않기 위해 훈련 세트에서 다시 떼어 낸 데이터 세트.

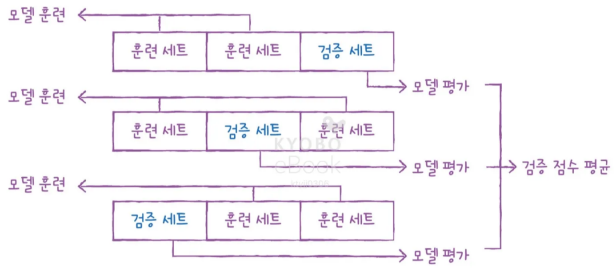
▼ 교차 검증

교차 검증 : 안정적인 검증 점수를 얻고 훈련에 더 많은 데이터 사용 가능

사이킷런 : cross_validate() 교차 검증 함수 사용. 훈련 세트 전체를 cross_validate() 함수에 전달

⇒ 훈련 세트를 여러 폴드로 나눈 다음 한 폴드가 검증 세트의 역할을 하고 나머지 폴드에서는 모델을 훈련.

⇒ 모든 폴드에 대해 검증 점수를 얻어 평균하는 방법



cross_validate() : 교차 검증 수행하는 함수

```
# 훈련 세트를 섞은 후 10-폴드 교차 검증 수행
# n_splits 매개변수는 몇 폴드 교차 검증을 할지 결정한다.
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']))
```

→ 첫번째 매개변수에 교차 검증 수행할 모델 객체를 전달, 두번째와 세번째 매개변수에 특성과 타깃 데이터 전달

→ scoring 매개변수 : 검증에 사용할 평가 지표 지정. 기본적으로 분류 모델은 ‘accuracy(정확도)’ 와 ‘r2(결정계수)’ 로 구성.

⇒ cross_validate() 는 훈련 세트를 섞어 폴드를 나누지 않음.

⇒ 교차 검증을 할 때 훈련 세트를 섞으려면 분할기를 지정해야 함.

▼ 하이퍼파라미터 튜닝 : 그리드 서치

모델이 학습할 수 없어서 사용자가 지정해야만 하는 파라미터 : 하이퍼파라미터

1. 탐색할 매개변수 지정
2. 훈련 세트에서 그리드 서치 수행하여 최상의 평균 점수가 나오는 매개변수 조합 찾음. 이 조합은 그리드 서치 객체에 저장됨.
3. 그리드 서치는 최상의 매개변수에서 전체 훈련 세트를 사용해 최종 모델을 훈련.

그리드 서치 :

하이퍼파라미터 탐색을 자동화해주는 도구. 탐색할 매개변수를 나열하면 교차 검증을 수행하여 가장 좋은 점수의 매개변수 조합 선택.

사이킷런 GridSearchCV 클래스 :

하이퍼파라미터 탐색과 교차 검증 한 번에 수행 , cross_validate() 함수 호출 필요 없음. (GridSearchCV 의 cv 매개변수 기본값 5)

▼ 랜덤 서치

랜덤 서치 :

매개변수 값의 목록을 전달하는 것이 아니라 매개변수를 샘플링 할 수 있는 확률 분포 객체 전달.

매개변수의 값이 수치일 때 값의 범위나 간격을 정하기 어려워 그리드 서치 수행 시간이 오래걸릴 때, 랜덤 서치 사용.

```
from scipy.stats import uniform, randint

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

array([8, 3, 8, 9, 7, 5, 4, 8, 5, 6])
```

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

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),
          }
```

▼ 랜덤포레스트

앙상블 학습 :

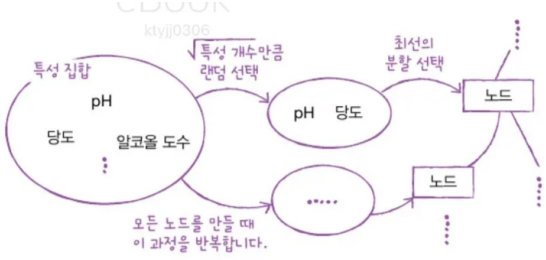
정형 데이터를 다루는 데 가장 뛰어난 성과를 내는 알고리즘

랜덤 포레스트 :

결정 트리를 랜덤하게 만들어 결정 트리(나무)의 숲을 만들고, 각 결정 트리의 예측을 사용해 최종 예측을 만듦.

부트스트랩 샘플 :

1000개의 샘플에서 100개 샘플을 뽑는다 하였을 때, 1개를 뽑고 다시 넣고 다시 뽑고...




```
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

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

엑스트라트리 :

부트스트랩 샘플을 사용하지 않고, 전체 훈련 세트를 사용.

```
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

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

▼ 그레이디언트 부스팅

깊이가 얕은 결정 트리를 사용하여 이전 트리의 오차를 보완하는 방법

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(random_state=42)
scores = cross_validate(gb, train_input, train_target, return_train_score=True, n_jobs

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

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

히스토그램 기반 그레이디언트 부스팅

: 노드의 특성을 나눈 뒤 사용. (입력에서 누락된 특성이 있어도 전처리가 필요 없음)

permutation_importance () 함수 :

특성 중요도를 계산

```
from sklearn.ensemble import HistGradientBoostingClassifier

hgb = HistGradientBoostingClassifier(random_state=42)
scores = cross_validate(hgb, train_input, train_target, return_train_score=True, n_job

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

-> 0.9321723946453317 0.8801241948619236

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]
```

그레이디언트 부스팅 알고리즘 구현 라이브러리 1 . XGBoost

```
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_job

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

-> 0.9558403027491312 0.8782000074035686
```

2. LightGBM

```
from lightgbm import LGBMClassifier

lgb = LGBMClassifier(random_state=42)
scores = cross_validate(lgb, train_input, train_target, return_train_score=True, n_job

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