


# ML\_06\_1. 트리 알고리즘

🕒 생성일	@2022년 6월 9일 오전 1:03
🏷️ 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

## 혼공머신: 5장 트리 알고리즘

로지스틱 회귀 모델을 분류 문제에 적용해보고, 다음으로 결정트리 모델을 분류에 적용해본다.

사용할 데이터는 알코올 도수, 당도, pH 값을 특성으로 갖고, 2개의 클래스(클래스가 0 이면 레드와인, 1이면 화이트 와인) 를 갖는다. 전형적인 binary classification 문제이다.

### 데이터 로드 및 전처리

판다스를 이용해 csv 데이터를 불러오고 살펴본다. 이후 사이킷런을 이용해 표준화 전처리 진행.

```
# 데이터 로드
import pandas as pd
wine = pd.read_csv("https://bit.ly/wine_csv_data")

# 데이터 탐색
wine.head()
wine.info()

>>> 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

wine.describe()
>>>    alcohol    sugar      pH      class
count  6497.000000  6497.000000  6497.000000  6497.000000
mean   10.491801   5.443235   3.218501   0.753886
std    1.192712    4.757804   0.160787   0.430779
min     8.000000    0.600000   2.720000   0.000000
25%    9.500000    1.800000   3.110000   1.000000
50%   10.300000    3.000000   3.210000   1.000000
75%   11.300000    8.100000   3.320000   1.000000
max   14.900000   65.800000   4.010000   1.000000

# 타겟으로 사용할 'class' column을 분리한다.
data = wine[['alcohol', 'sugar', 'pH']].to_numpy()
target = wine['class'].to_numpy()

# 훈련 데이터, 테스트 데이터 분리
# 테스트 데이터의 비율을 전체 데이터의 20%로 지정
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
)

# 위의 wine.describe() 를 보면 특성별로 값의 스케일 차이가 크다.
# 표준화 전처리
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

```
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

## 로지스틱 회귀 모델 생성 및 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
```

테스트 점수가 다소 낮게 측정된다.

## 결정트리 (Decision Tree)

결정 트리는 다음과 같이 질문을 통해서 클래스를 분류해나간다.



사이킷런에서는 sklearn.tree 라이브러리에 결정트리 분류 클래스를 구현해놓았다. 위에서 전처리한 데이터셋을 가지고 트리 모델을 학습시켜 보자.

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)

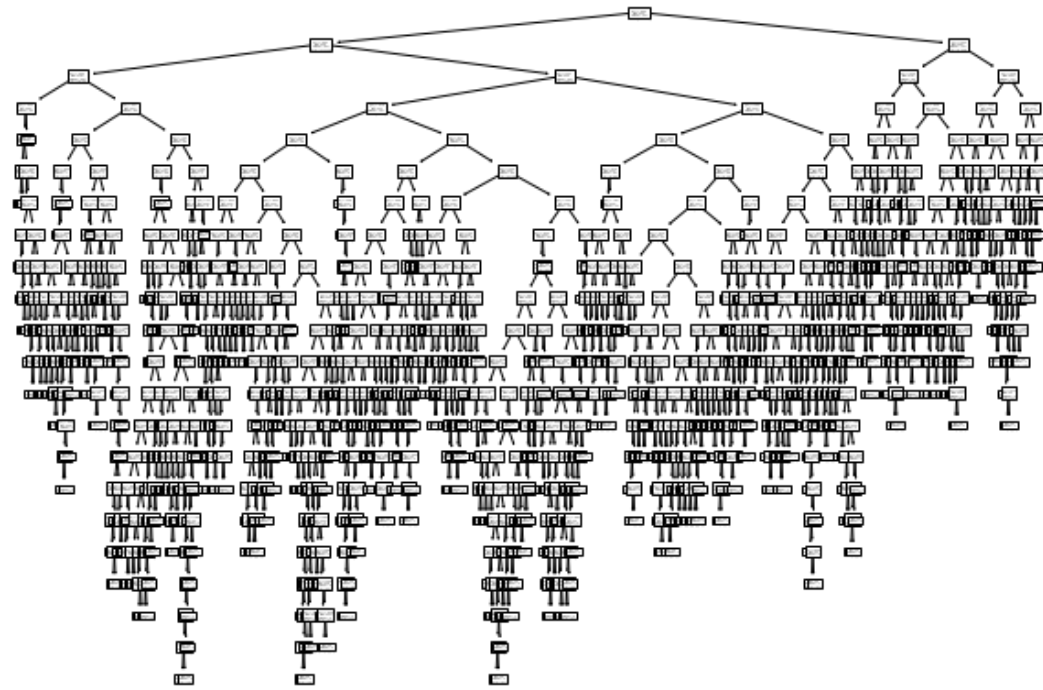
dt.fit(train_scaled, train_target)
print(dt.score(train_scaled, train_target))
print(dt.score(test_scaled, test_target))

>>> 0.996921300750433
      0.8592307692307692
```

로지스틱 회귀 모델 보다 확실히 스코어가 높게 측정된다. 하지만 훈련 세트 점수에 비해 테스트 성능이 낮은, 과대적합 경향이 강하다. 이후 결정 트리 모델에서 과대적합을 줄이기 위한 방향으로 진행 예정.

전체적인 트리의 구조를 확인하고 싶다면 다음의 코드를 실행시키면 된다. plt.show() 결과는 코드 아래의 이미지와 같다.

```
# 결정 트리 모델 객체 그래프
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(10,7))
plot_tree(dt)
plt.show()
```



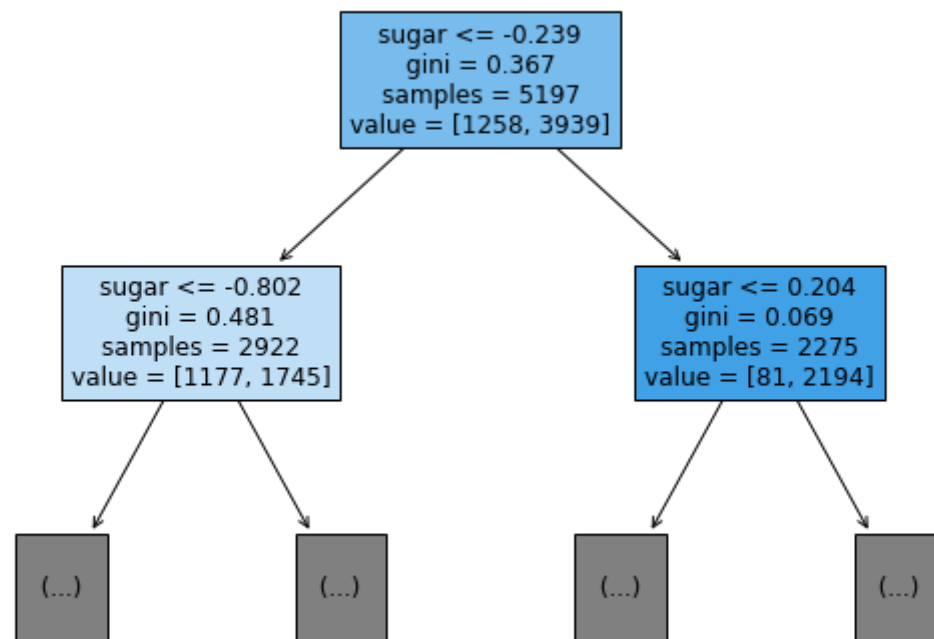
위의 이미지를 통해 결정 트리의 몇 가지 구조적 특징과 개념을 이해할 수 있다.

- 결정 트리는 위에서부터 아래 방향으로 진행한다.(top-down)
- node 는 훈련 데이터의 특성에 대한 테스트를 표현한다.
- branch 는 테스트의 결과(boolean; True or False) 를 나타내며 일반적으로 하나의 노드는 2 개의 branch를 가진다.
- 맨 위의 node 를 root node, 맨 아래 끝에 달린 node를 leaf node 라고 한다.

트리 구조는 부분적으로 확인할 수 있다. 다음의 코드를 실행한 결과는 그 아래 이미지이다.

```
'''
트리의 깊이를 제한해서 출력
max_depth=1 --> 루트 노드와 그 바로 아래의 노드 층까지만 출력
filled=True --> 클래스에 맞게 노드 색 변화
feature_names --> 특성의 이름 전달.
'''

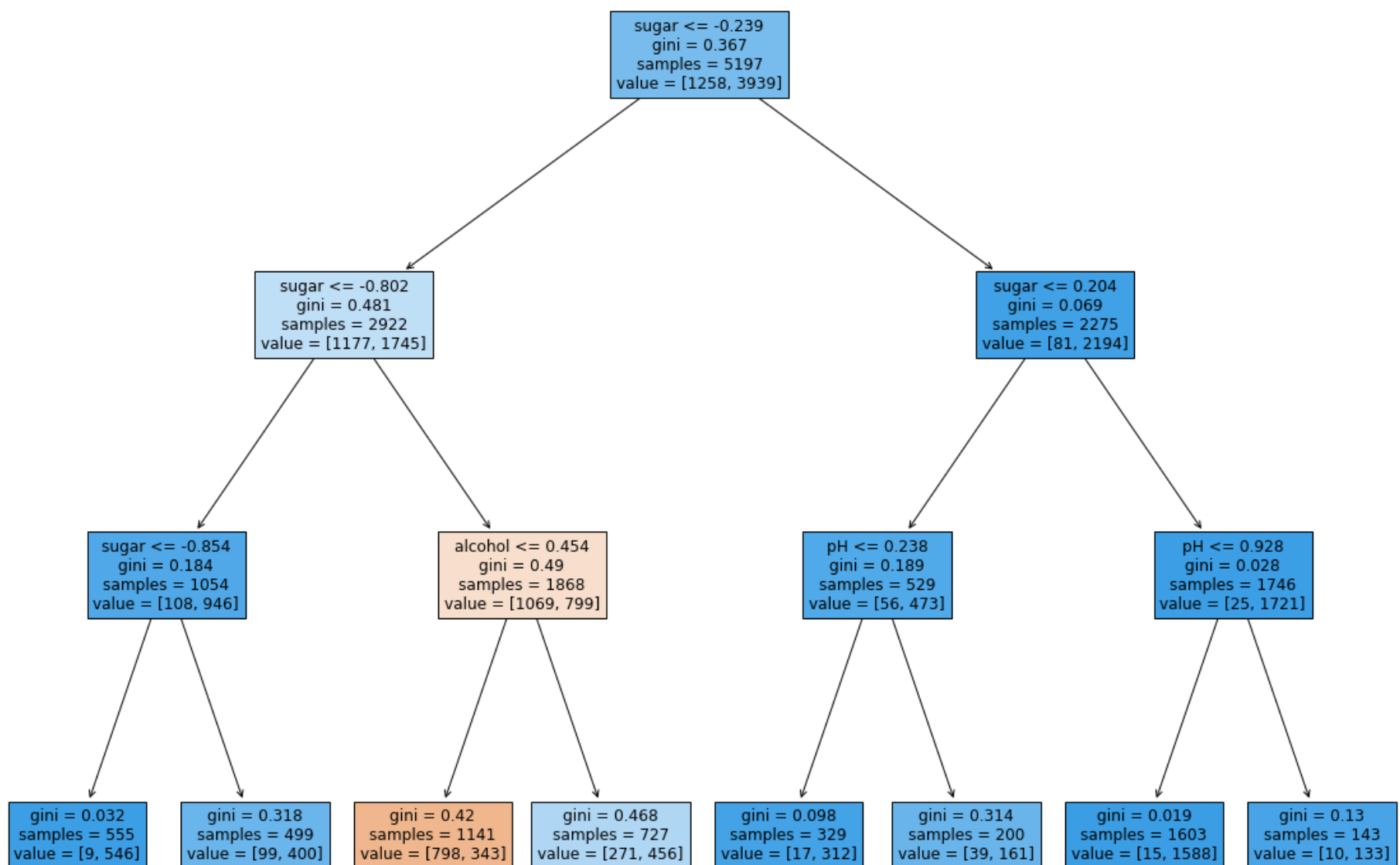
plt.figure(figsize=(10,7))
plot_tree(dt, max_depth=1, filled=True, feature_names=['alcohol', 'sugar', 'pH'])
plt.show()
```



결정 트리에서는 자라날 수 있는 트리의 최대 깊이(max\_depth)를 제한할 수 있다. 아래 코드는 트리의 최대 깊이를 3으로 지정해서 분기가 어떻게 진행되는지 시각화한 것이다.

```
# 트리 객체 생성
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(train_scaled, train_target)

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



결정트리 모델에서는 어떤 특성이 분류에 어느 정도 영향을 끼치는지 중요도를 계산할 수 있다.

```
# 특성 중요도 계산
print(dt.feature_importances_)
>>> [0.12345626  0.86862934  0.0079144 ]
```

## 검증 세트(validation set)

현재 train set : test set = 0.8 : 0.2 로 나뉘어 있다. train set : valid set : test set = 0.6 : 0.2 : 0.2 로 구성하여 validation set을 모델 평가에 활용해보자. 이번에는 표준화 전처리를 진행하지 않는다.(사실, 트리모델에 input 데이터는 스케일 조정이 필요하지 않다.)

```
# 원본 데이터셋을 따로 저장하지 않았기에 다시 로드한다.
import pandas as pd
wine = pd.read_csv('https://bit.ly/wine_csv_data')

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

# 우선 훈련 세트와 테스트 세트를 나눠 준다.
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)

# validation 세트 구성
sub_input, val_input, sub_target, val_target = train_test_split(
    train_input, train_target, test_size=0.2, random_state=42)

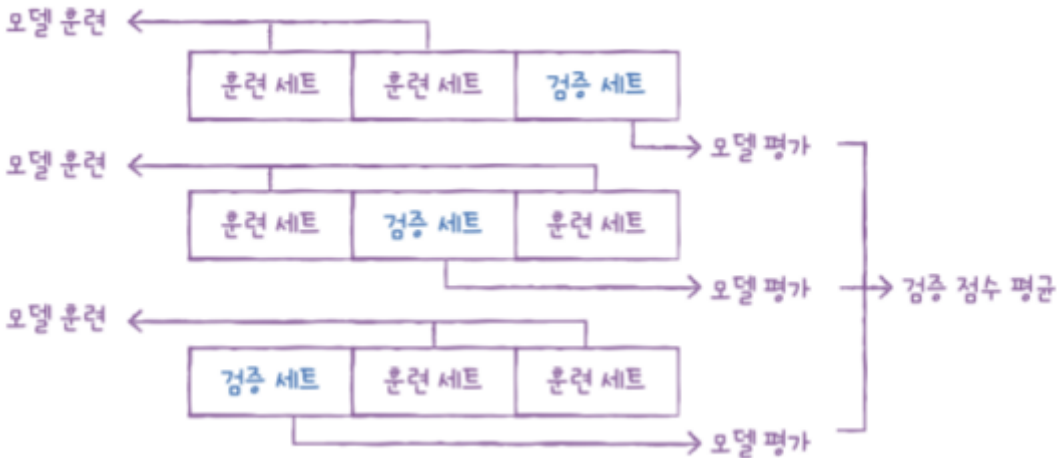
# 모델 생성 및 평가
# 모델 평가에 테스트 셋이 아닌 valid set을 사용.
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(sub_input, sub_target)
print(dt.score(sub_input, sub_target))
print(dt.score(val_input, val_target))

>>> 0.9971133028626413
      0.864423076923077
```

## 교차 검증(cross-validation)

교차 검증은 검증 세트를 떼어 내어 평가하는 과정을 여러 번 반복한다. 그 다음 이 점수를 평균하여 최종 검증 점수를 얻는다. 다음은 교차 검증 방법 중 k-폴드 교차 검증에 대한 내용이다.

ex. 3-폴드 교차 검증



훈련 세트를 k 개 부분으로 나눠서 교차 검증을 수행하는 것을 'k-폴드 교차 검증' 이라고 한다. 일반적으로 5-폴드 나 10폴드 교차 검증을 많이 사용한다.

교차 검증을 통해 전체 데이터의 80-90% 까지 훈련에 사용할 수 있다. k-폴드 교차 검증 방식을 사용하면, 검증 세트가 줄어들지만 각 폴드에 서 계산한 검증 점수를 평균하기 때문에 신뢰할 만한 점수로 생각할 수 있다.

훈련 데이터의 양이 적은 상황에서 검증 데이터를 따로 분리시키기 어렵다면, k-폴드 교차 검증 방식은 고려해볼만 하다.

```
# 사이킷런 cross_validate() 교차 검증 함수 사용하기
from sklearn.model_selection import cross_validate
scores = cross_validate(dt, train_input, train_target) # 순서대로 (모델, 검증세트를 떼어내지 않은 훈련 세트, target)
print(scores)

>>> {'fit_time': array([0.0112164 , 0.0107832 , 0.00883532, 0.00893021, 0.00702477]),
'score_time': array([0.00123191, 0.0015707 , 0.00109911, 0.00084567, 0.00076032]),
'test_score': array([0.86923077, 0.84615385, 0.87680462, 0.84889317, 0.83541867])}

# 위의 'test_score' 를 평균하여 교차 검증의 최종 점수 산출
print(np.mean(scores['test_score']))
>>> 0.855300214703487
```

만약 교차 검증을 할 때 훈련 세트를 에포크마다 섞으려고 한다면 splitter 를 지정해야 한다. cross\_validate() 메서드는 회귀 모델일 경우 KFold splitter 를 사용하고 분류 모델일 경우 타깃 클래스를 골고루 나누기 위해 StratifiedKFold 를 사용한다.

```
# StratifiedKFold 분할기를 사용한 교차 검증
from sklearn.model_selection import StratifiedKFold
scores = cross_validate(dt, train_input, train_target, cv=StratifiedKFold())
```

---

## ISLR : 8. Tree-Based Methods

트리 기반 모델은 해석하는데 단순하고 유용하지만, 관측은 다른 지도학습 알고리즘에 비해 경쟁력 있지 않다. 이 챕터에서는 기본적인 트리 기반 모델을 알아본 다음, bagging, random forest, boosting 과 같은 알고리즘을 추가적으로 공부할 예정이다. 이러한 알고리즘들은 여러 트리를 조합해서 예측에 사용하는데, 해석하는데 어렵지만, 드라마틱한 성능 향상을 기대할 수 있다.

-내용 수정 중 -

---

## 핸즈온 : 6. 결정 트리

핸즈온 6장 에서는 결정 트리의 훈련, 시각화, 예측 방법에 대해 먼저 살펴보고, 사이킷런의 CART 훈련 알고리즘을 설명한다. 추가적으로 트리에 규제를 가하는 방법과 회귀 문제에 적용하는 법, 결정 트리의 제약 사항에 관해 다룬다.

핸즈온에서 역시 혼공머신과 동일한 방식으로 사이킷런에서 결정트리분류 클래스를 객체 생성한 뒤, plot\_tree 를 통해 시각화 한다. 아래 코드에서 데이터는 iris 를 사용했다.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

# 데이터 준비
iris = load_iris()
```

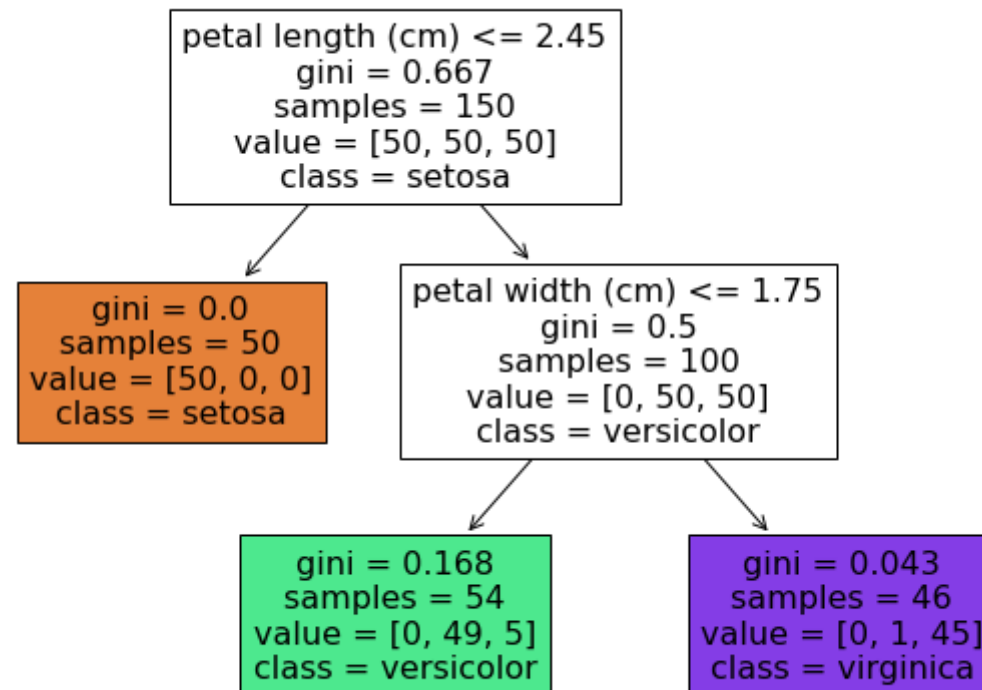
```

X = iris.data[:, 2:] # 꽃잎 길이와 너비
y = iris.target

# 결정 트리 학습
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)

# 시각화
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(10,7))
plot_tree(tree_clf, max_depth=2, filled=True,
          feature_names=iris.feature_names[2:],
          class_names=iris.target_names)

```



위의 이미지를 보면 트리에서 노드가 특정 조건을 기준으로 분기되는 것을 알 수 있다. 특이한 점은 gini 라는 인덱스가 있는데 노드의 gini 속성은 ‘불순도’를 의미하며 트리 알고리즘의 중요한 요소이다. 예를 들어 한 노드의 모든 샘플이 하나의 클래스에 속해있다면 gini = 0 이 된다. 깊이 = 1 의 노드를 살펴보면 오른쪽 노드는 gini = 0.5 이다. 이것을 계산하는 방식은 다음의 식을 따른다.

$$1 - (0/100)^2 - (50/100)^2 - (50/100)^2 = 0.5$$

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

- 이 식에서  $p_{i,k}$ 는  $i$ 번째 노드에 있는 훈련 샘플 중 클래스  $k$ 에 속한 샘플의 비율입니다.

## CART 훈련 알고리즘

분류에 대한 CART 비용 함수는 다음과 같다.

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

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

먼저 훈련 세트를 하나의 특성  $k$  의 임계값  $t_k$  를 사용해 두 개의 서브셋으로 나눈다. 이때  $k$ ,  $t_k$  를 고르는 기준은 가장 순수한 서브셋으로 나눌 수 있는  $(k, t_k)$  짝을 찾는 것이다. 이 때 최소화해야 하는 비용 함수는 위의 식을 따른다.

서브셋을 계속해서 나누는 것을 반복하며, 미리 설정해둔 최대 깊이가 되면 중지하거나 불순도를 줄이는 분할을 찾을 수 없을 때 중지한다.

## 불순도 - 지니, 엔트로피

지니 불순도가 아닌 엔트로피 불순도를 사용할 수도 있다. 트리 클래스에서 criterion 매개변수를 'entropy' 로 지정하면 된다. 어떤 세트가 한 클래스의 샘플만 담고 있다면 엔트로피가 0이다.

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

지니 불순도와 엔트로피 불순도는 실제로 큰 차이가 없다. 지니 불순도가 조금 더 계산이 빠르다.

## 규제 매개변수

결정 트리는 훈련 데이터에 대한 제약 사항이 거의 없다. 제한을 두지 않으면 트리가 훈련 데이터에 아주 가깝게 맞추려고 해서 대부분 과대적합되기 쉽다. 결정트리는 모델 파라미터가 없는 것이 아니라 훈련되기 전에 파라미터 수가 결정되지 않는 것 뿐이다.

과대적합을 피하기 위해 학습할 때 결정 트리의 자유도를 제한할 필요가 있다. 예를 들어 트리의 성장에 제한을 주는, max\_depth 를 설정하는 것이다. 다른 규제 방식을 정리하면 다음과 같다.

- min\_samples\_split : 분할되기 위해 노드가 가져야 하는 최소 샘플 수
- min\_samples\_leaf : 리프 노드가 가지고 있어야 할 최소 샘플 수
- max\_features : 분할에 사용할 특성의 최대 수

## 가지치기

제한 없이 결정 트리를 훈련시키고 불필요한 노드를 제거하는 알고리즘도 있다. 순도를 높이는 것이 통계적으로 큰 효과가 없다면 리프 노드 바로 위의 노드는 불필요할 수 있다. 대표적으로 카이 제곱 검정 과 같은 통계적 검정을 사용하여 우연히 향상된 것인지 추정한다. 이 확률을 p-value 라고 부르며 어떤 임계값(통상적으로 5%)보다 높으면 그 노드는 불필요한 것으로 간주되고 그 자식 노드는 삭제된다. 가지치기는 불필요한 노드가 모두 없어질 때까지 계속된다.

## 회귀

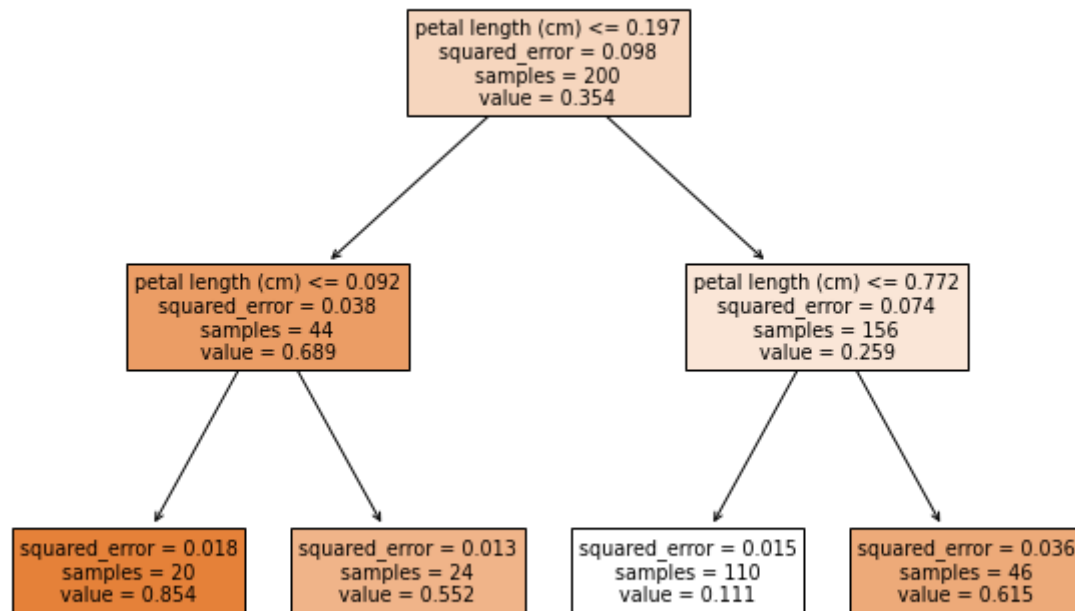
결정 트리는 회귀문제에도 사용할 수 있다.

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)

from sklearn.tree import plot_tree
plt.figure(figsize=(10, 7))
plot_tree(tree_reg, max_depth=2, filled=True,
          feature_names=iris.feature_names[2:],
          class_names=iris.target_names)
```





분류 문제와는 달리 클래스가 아니라 value 값을 산출한다.

회귀 문제에서 CART 알고리즘은 훈련 세트를 불순도를 최소화하는 방향으로 분할하는 대신 MSE를 최소화하도록 분할한다. 회귀를 위한 CART 비용 함수는 다음과 같다.

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

여기서

$$\begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

## 인공지능 및 기계학습 개론

hypothesis : rule based algorithm → Decision Tree

ex. credit approval dataset

### Entropy

Entropy = index that check a better attribute → reduce the uncertainty

About random variables(=Features), higher entropy means more uncertainty.

Conditional entropy : We are interested in the entropy of the class given a feature variable.

-내용 작성 중-