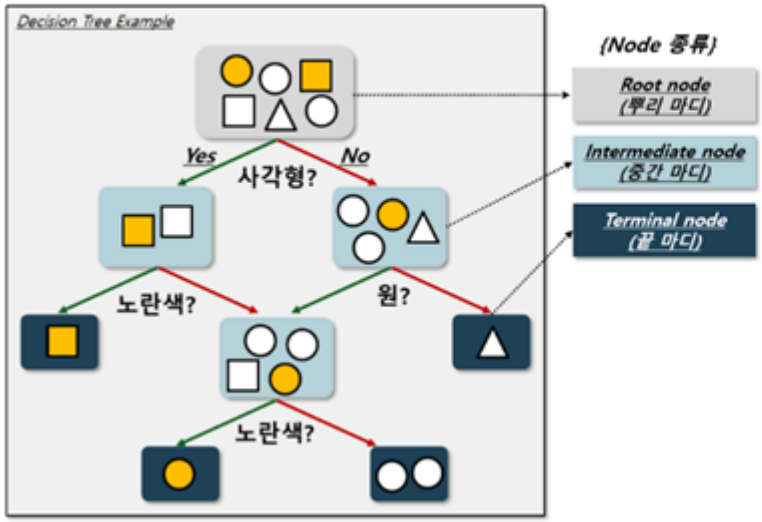
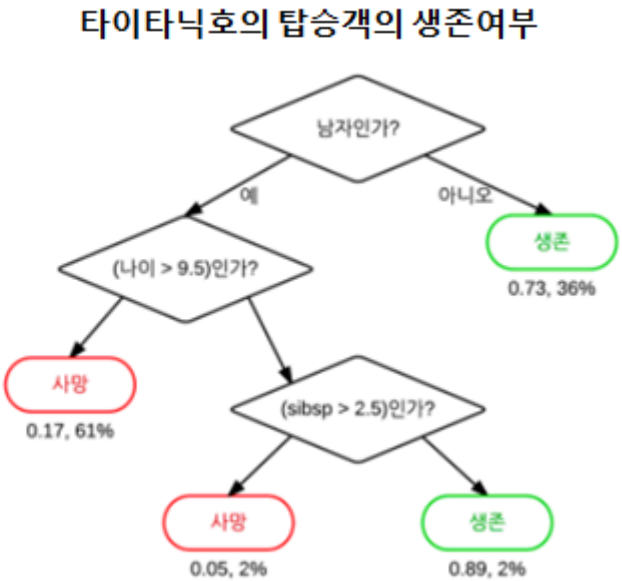
결정트리(Decision Tree, 의사결정트리, 의사결정나무)

- 의사결정트리는 일련의 분류 규칙을 통해 데이터를 분류, 회귀하는 지도 학습 모델 중 하나
  - 종속변수의 형태에 따라 분류와 회귀 문제로 나뉨
    - 종속변수가 범주형일 경우 Decision Tree Classification으로 분류를 진행하고, 종속변수가 연속형일 경우 Decision Tree Regression으로 회귀를 진행함
  - 결과 모델이 Tree 구조를 가지고 있기 때문에 Decision Tree라는 이름을 가짐
  - if~else와 같이 특정 조건을 기준으로 O/X로 나누어 분류/회귀를 진행하는 tree구조의 분류/회귀 데이터마이닝 기법





위의 예시에서 모양 or 색 중에서 무엇을 먼저, 어떤 기준으로 나눠야 할까?

- 답은 불순도가 낮아지는 방향으로 나눠야 함
    - 결정 트리의 기본 아이디어는, Leaf Node가 가장 섞이지 않은 상태로 완전히 분류되는 것, 즉 복잡성(entropy)이 낮도록 만드는 것임 링크 텍스트
- 방법 - 불순도를 수치화한 지표(정보량의 기댓값)로 엔트로피(Entropy), 지니계수(Gini Index) 링크 텍스트 등이 있음
    - 불순도를 엔트로피로 계산한 알고리즘 - ID3 알고리즘
    - 불순도를 지니계수로 계산한 알고리즘 - CART 알고리즘

정보이론(Information theory)과 정보량(Information)

- 정보이론
    - 정보의 양과 통신에서의 정보 전송량 등을 측정하고, 처리하는 수학적인 이론
- 정보량(Information)
    - 정보이론에서 가장 기본적인 개념
    - 정보량은 어떤 사건이 얼마나 놀라운가에 따라서 결정됨
        - 더 드문 사건일수록 정보량이 크며, 반대로 더 일어날 가능성이 높은 사건일수록 정보량이 작아짐

- 예를들면 로또에 당첨되지 않았어~ 라는 말은 굉장히 흔한 일이기 때문에 정보량이 굉장히 작음
- 하지만 로또에 당첨되었다는 것은 굉장히 흔하지 않은 일이기 때문에 정보량이 많은 것임

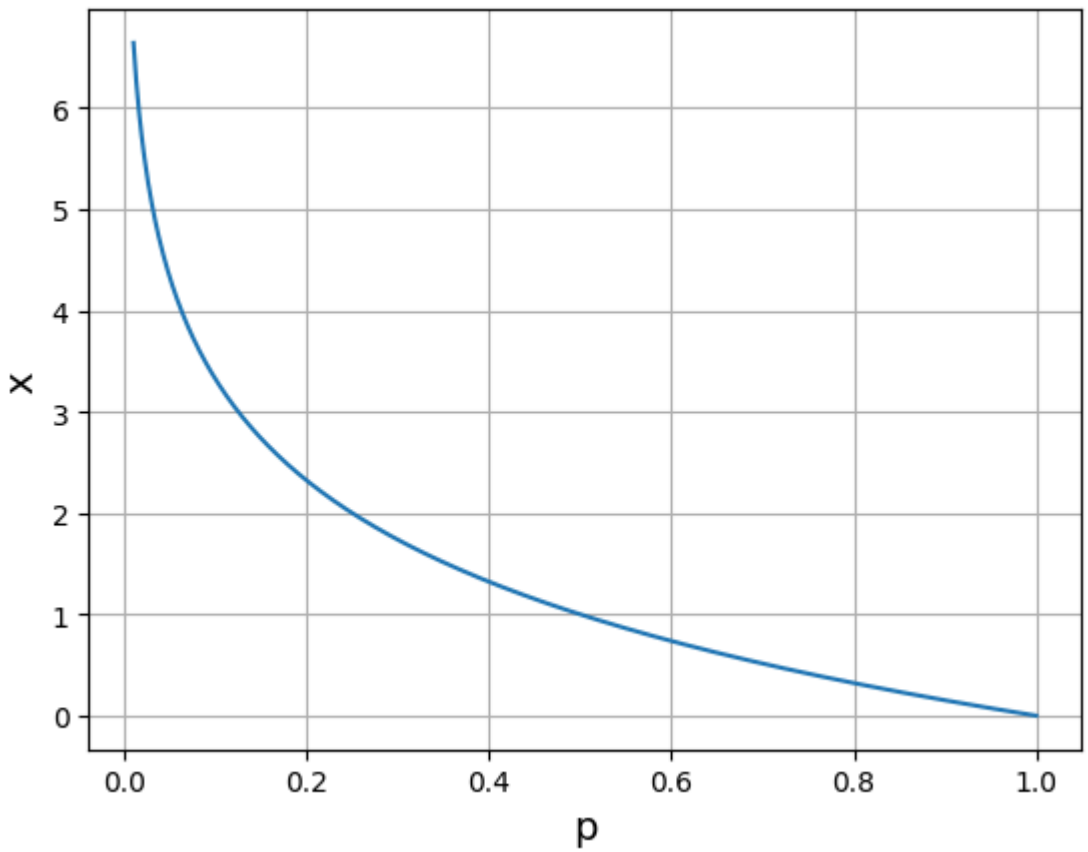
- 정보량을 수학적으로 나타낼 때는 엔트로피(Entropy)라는 개념을 사용

정보량이란 어떤 사건이 가지고 있는 정보의 양을 의미하며, 식으로 다음과 같다.

$$I(x) = log_2 \frac{1}{p(x)}$$

p(x) : 사건 x가 발생할 확률

```
# 정보량
#사건 x가 발생할 확률을 x축, 정보량을 y축으로 그래프를 그리면 다음과 같음
#사건 x가 발생할 확률이 증가할 수록, 정보량은 0에 수렴함
#즉, "흔하게, 자주 발생하는 사건일수록 그닥 많은 정보를 가지고 있지 않다." 라고 해석할 수 있음
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

p = np.linspace(0.01, 1, 1000) # 0일 때 로그는 발산하므로 0.01부터 시작!
y = -np.log2(p)

plt.plot(p, y)

plt.xlabel('p', size = 14)
plt.ylabel('x', size = 14)
plt.grid()

plt.show()
```



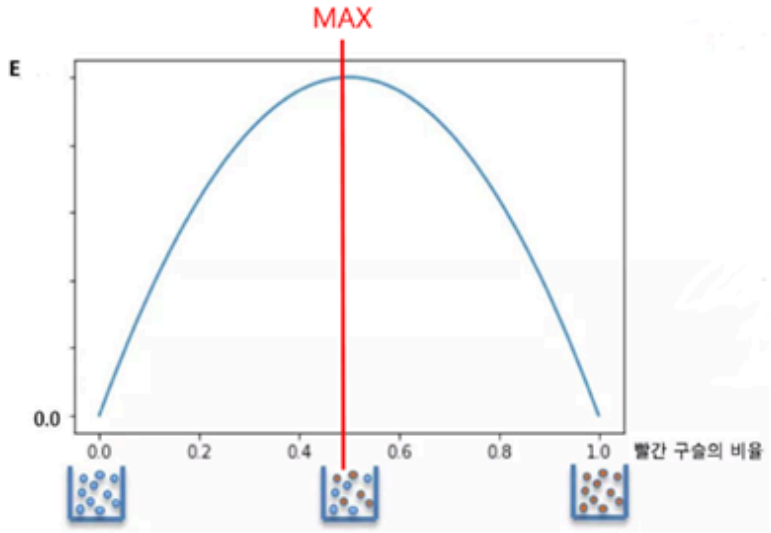
불순도(Impurity)

- 항아리 세개(1번과 3번항아리를 오롯이 파란공, 빨간공으로만 채워져 있으며, 2번 항아리는 빨간공과 파란공이 정확히 반반 섞여있을때)
  - 1번과 3번 항아리는 순도 100%라 할 수 있으며, 2번 항아리는 불순도가 높은 상태라 할 수 있음

항아리 1.

항아리 2.

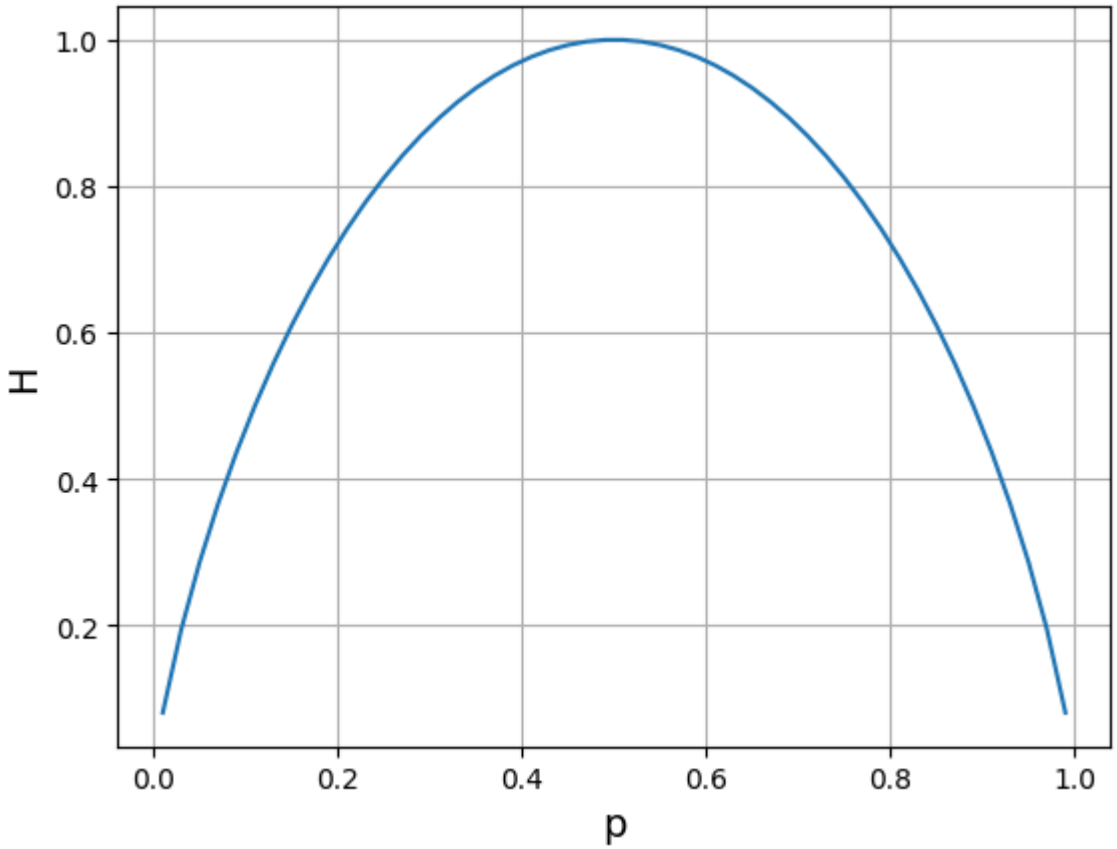
항아리 3.



- ✓ 항아리에 순도가 100% (한 종류의 공만 있는 상태 = 분류하기 좋은 상태)일 때 Entropy는 0이며, 두 공이 정확히 반반 섞여 있을 때 (불순한 상태 = 분류하기 어려운 상태) Entropy가 최대값을 가짐
- ✓ 즉, 불순한 상태일 수록 Entropy는 큰 값을 가지며, 불순도가 클 수록 분류하기 어려움
- ✓ **이에 따라, Entropy를 작게하는 방향으로 가지를 뻗어나가며 의사결정나무를 키워나가는 것이 ID3 알고리즘의 핵심이라 할 수 있음!!!!**

```
# 베르누이 분포의 엔트로피
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

p = np.linspace(0.01, 0.99) # 0일 때 로그는 발산하므로 0.01 ~ 0.99로 제한

H = -p * np.log2(p) - (1 - p) * np.log2(1 - p) # 베르누이 분포의 엔트로피

```

```
plt.plot(p, H)

plt.xlabel('p', size = 14)
plt.ylabel('H', size = 14)
plt.grid()

plt.show()
```



불순도 함수(Gini, Entropy)

- **지니 지수(Gini)**
  - 공식

$$I(A) = 1 - \sum_{k=1}^{m} p_k^2$$

    - **지니 지수의 최대값은 0.5이다.**

  - 범주 안에 빨간색 점 10개, 파란색 점이 6개 있을 때의 계산 예제입니다.

$$\begin{aligned} I(A) &= 1 - \sum_{k=1}^{m} p_k^2 \\ &= 1 - \left(\frac{6}{16}\right)^2 - \left(\frac{10}{16}\right)^2 \\ &\approx 0.47 \end{aligned}$$

- **엔트로피 지수(Entropy)**
  - 공식

$$E = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

  - 범주 안에 빨간색 점 10개, 파란색 점이 6개 있을 때의 계산 예제입니다.

$$\begin{aligned} E(A) &= -\sum_{k=1}^{m} p_k \log_2(p_k) \\ &= -\frac{6}{16}\log_2\left(\frac{6}{16}\right) - \frac{10}{16}\log_2\left(\frac{10}{16}\right) \\ &\approx 0.95 \end{aligned}$$

정보획득(Information gain)

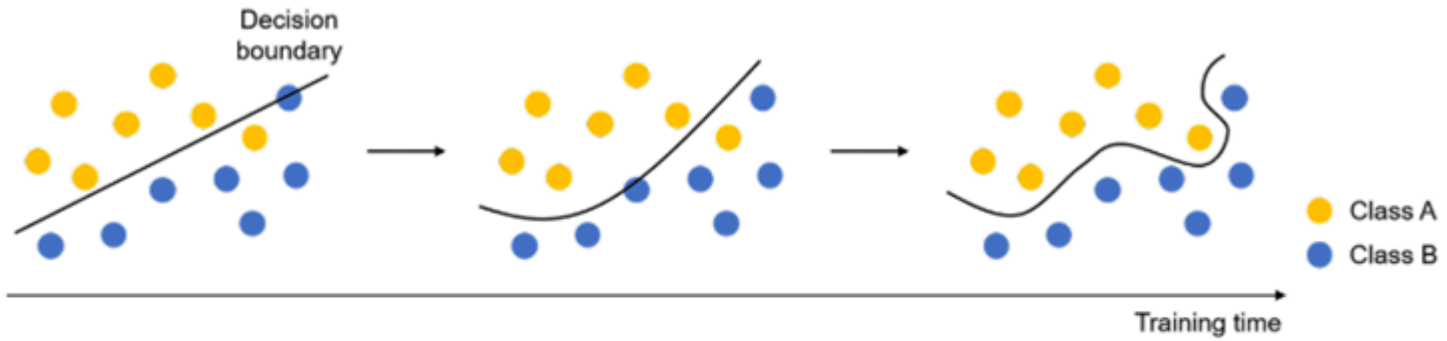
- 분기 이전의 불순도와 분기 이후의 불순도의 차이
  - 불순도가 1인 상태에서 0.7인 상태로 바뀌었다면 정보 획득(information gain)은 0.3임

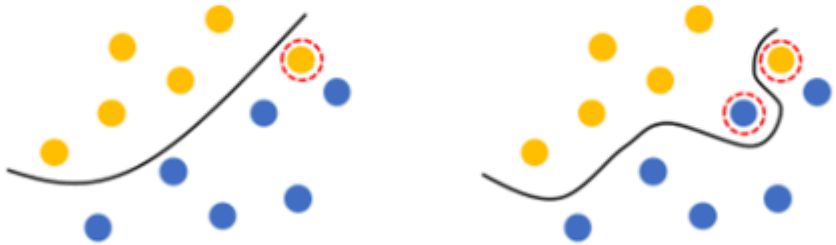
결정트리 구성 단계

- Root 노드의 불순도 계산
- 나머지 속성에 대해 분할 후 자식노드의 불순도 계산
- 각 속성에 대한 Information Gain 계산 후 Information Gain(Root 노드와 자식노드의 불순도 차이)이 최대가 되는 분기조건을 찾아 분기
- 모든 leaf 노드의 불순도가 0이 될때까지 2, 3을 반복 수행

과적합(Overfitting)과 Validation Dataset의 개념

- 과적합 문제(overfitting problem)
  - train에 과하게 적합되면 예측 모델이 train에서는 더 낮은 예측 오차를 보이지만 실제 test에서는 더 높은 예측 오차를 보이는 것
    - 머신 러닝 모델은 train에 대한 손실 함수(loss function)가 작아지도록 학습을 진행하기 때문에 분류 문제의 경우 학습이 진행될수록 모델의 decision boundary는 train에 적합(fitting)됨
    - 그러나 모델이 train에 너무 과하게 적합되면 모델이 데이터에 내재된 어떠한 구조나 패턴을 일반화한 것이 아니라, train에만 한정되는 정보를 그대로 외운 것과 같을 수 있기 때문에 올바른 학습 결과인지는 생각해볼 필요가 있음
      - 과적합 문제는 대부분의 응용에서 발생하며 학습 데이터가 적거나 문제가 어려울수록 과적합의 정도가 심해짐





a. Best prediction model b. Overfitted prediction model

. Training dataset에 적당히 적합된 최적 예측 모델 (a)과 과대적합이 발생한 예측 모델 (b)

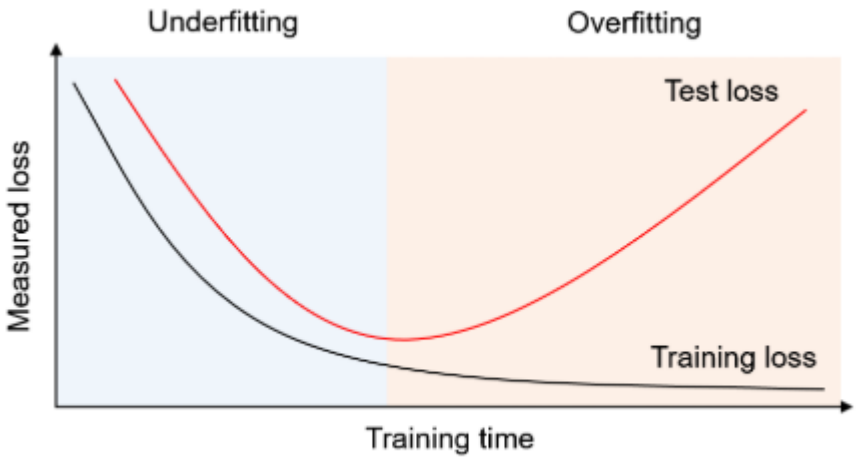
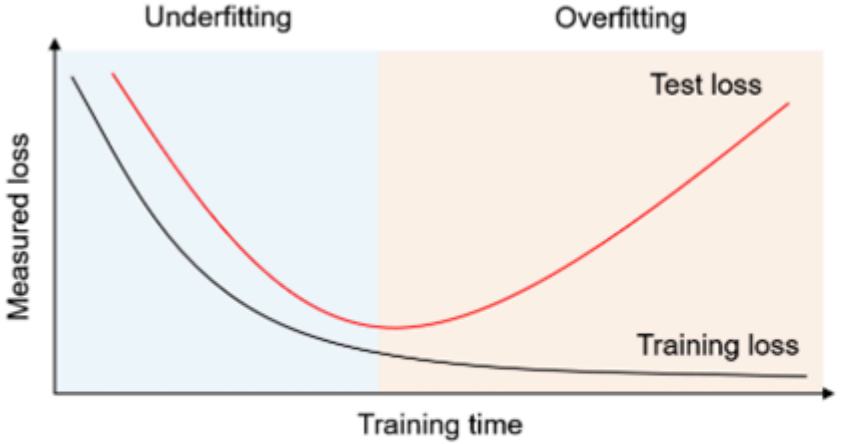


그림 3. 예측 오차의 관점에서 보는 과소적합과 과대적합



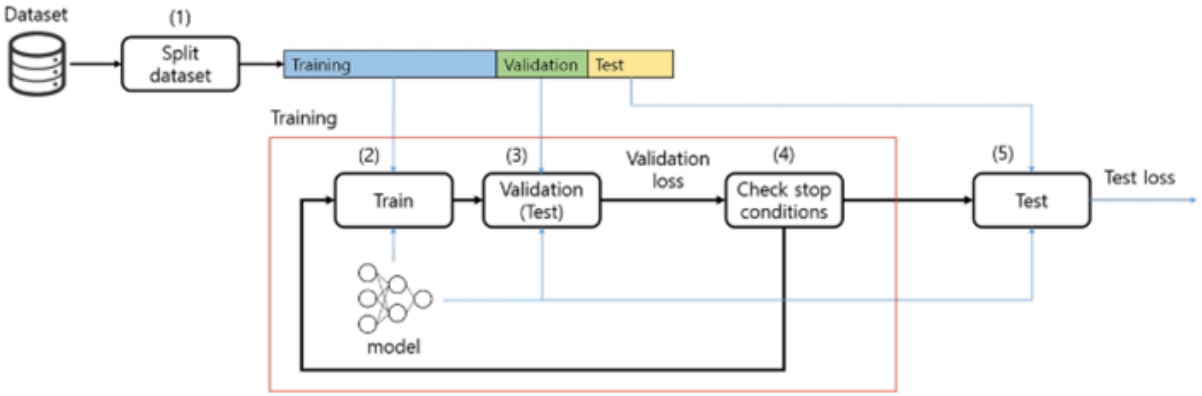
. 예측 오차의 관점에서 보는 과소적합과 과대적합

- **구간 A:** 학습 오차와 테스트 오차가 같이 감소하는 구간 (과소적합, underfitting).
- **구간 B:** 학습 오차는 감소하지만, 테스트 오차는 증가하는 구간 (과대적합, overfitting).

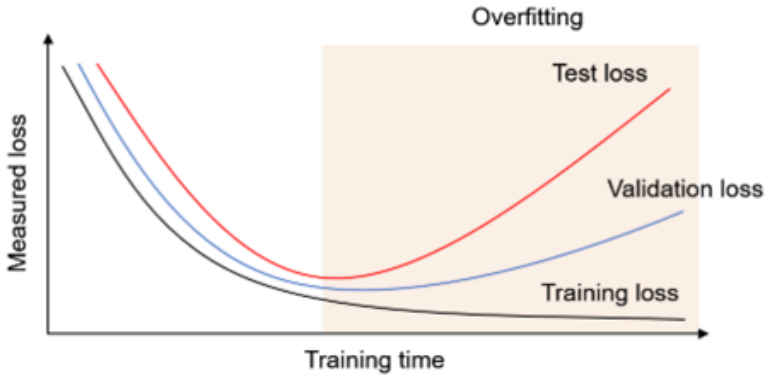
우리의 목적은 학습을 통해 예측모델의 과소적합된 부분을 제거해나가면서 과대적합이 발생하기 직전에 학습을 멈추는 것임!!!!

- Validation Dataset

  - 모델의 학습 과정에 참조되어 과대적합이 발생했는지를 판별하기 위해 사용되는 별도의 데이터셋
  - 일반적으로 실제 머신러닝 개발에서는 Validation Dataset을 별도로 구축하기보다는 train에서 일부 데이터를 추출하여 구축함



Validation dataset을 이용한 모델의 학습 과정



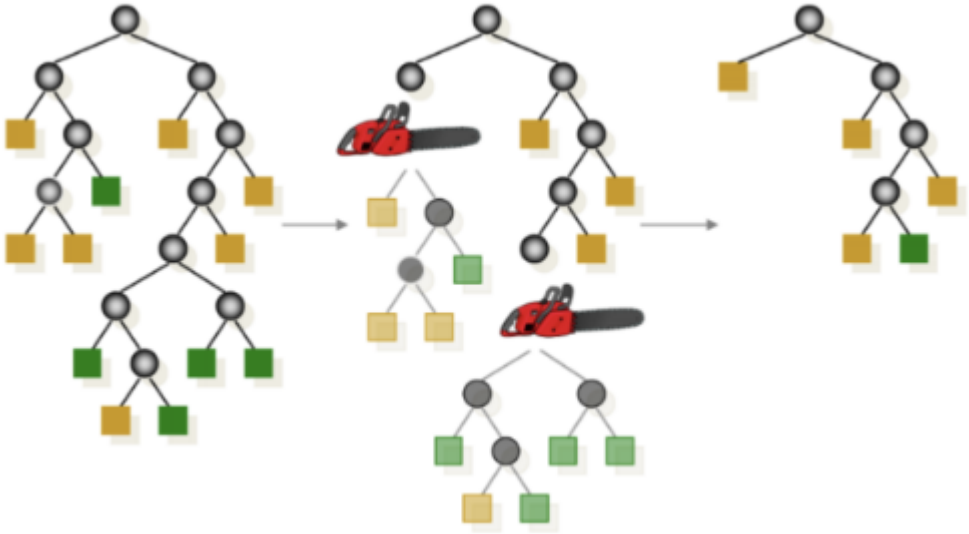
Validation loss 기반의 과대적합 판별

일반화

- 결정트리 구성방법을 사용하여 트리를 형성하게 되면, leaf 노드가 순도 100%의 한가지 범주만을 가지게 되는 Full tree(최대 트리)를 형성하게 됨
- 이러한 최대 트리는 새로운 데이터에 적용할 때 과적합 문제(Overfitting)가 발생하여 일반화 성능이 떨어지게 됨
- 따라서 형성된 결정트리에 대해 가지치기(Pruning)를 수행하여 일반화 성능을 높힘

가지치기(pruning)

- 최대트리로 형성된 결정트리의 특정 노드 밑의 하부 트리를 제거하여 일반화 성능을 높히는 것을 의미함(오버피팅을 막기위해 사용)
- 더 많은 가지가 생기지 않도록 최대 깊이, leaf 노드의 최대 개수, 한 노드가 분할하기 위한 최소 데이터 수 등의 제한이 가능함



가지치기의 비용함수

- 의사결정나무는 비용함수를 최소로 하는 분기를 찾아내도록 학습됨
  - CC(T) = Err(T) + α × L(T)
    - CC(T) : 의사결정나무의 비용 복잡도(오류가 적으면서 terminal node 수가 적은 단순한 모델일 수록 작은 값)

- ERR(T) : 검증데이터에 대한 오분류율
- L(T) : terminal node의 수(구조의 복잡도)
- Alpha : RR(T)와 L(T)를 결합하는 가중치(사용자에 의해 부여됨, 보통 0.01~0.1의 값을 씀)

결정트리의 장점

- 데이터의 전처리(정규화, 결측치, 이상치 등)를 하지 않아도 됨
- 수치형과 범주형 변수를 한꺼번에 다룰 수 있음

결정트리의 한계

- 만약 샘플의 사이즈가 크면 효율성 및 가독성이 떨어짐
- 과적합으로 알고리즘 성능이 떨어질 수 있음
    - 이를 극복하기 위해서 트리의 크기를 사전에 제한하는 튜닝이 필요함
- 한 번에 하나의 변수만을 고려하므로 변수간 상호작용을 파악하기가 어려움
- 결정트리는 Hill Climbing 방식 및 Greedy 방식을 사용하고 있음
    - 일반적인 Greedy 방식의 알고리즘이 그렇듯이 이 방식은 최적의 해를 보장하지는 못함
        - 언덕 오르기 방법(hill climbing method) - 현재 노드에서 확장 가능한 이웃노드들 중에서 휴리스틱에 의한 평가값이 가장 좋은 것 하나만을 선택해서 확장해 가는 탐색 방법

    - 그리디 알고리즘(탐욕법, Greedy Algorithm) - 최적의 값을 구해야 하는 상황에서 사용되는 근시안적인 방법론으로 '각 단계에서 최적이라고 생각되는 것을 선택' 해 나가는 방식으로 진행하여 최종적인 해답에 도달하는 알고리즘(이때, 항상 최적의 값을 보장하는것이 아니라 최적의 값의 '근사한 값'을 목표로 하고 있음)
- 약간의 차이에 따라 (레코드의 개수의 약간의 차이) 트리의 모양이 많이 달라질 수 있음
- 두 변수가 비슷한 수준의 정보력을 갖는다고 했을 때, 약간의 차이에 의해 다른 변수가 선택되면 이후의 트리 구성이 크게 달라질 수 있음

    - 이같은 문제를 극복하기 위해 등장한 모델이 바로 랜덤포레스트임
    - 같은 데이터에 대해 의사결정나무를 여러 개 만들어 그 결과를 종합해 예측 성능을 높이는 기법임

기상조건에 따른 테니스경기 참가여부를 ID3 알고리즘으로 분류

날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

Approach 1) 분할 전, 목표 속성인 참가여부에 대한 엔트로피 계산

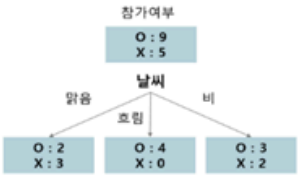
$$E(\text{경기}) = -\frac{9}{14}log_2(\frac{9}{14}) - \frac{5}{14}log_2(\frac{5}{14}) = 0.940$$

Approach 2) 각 속성에 대해 분할 후 엔트로피 계산

2-1) 날씨

$$E(\text{경기}|\text{날씨}) = p(\text{맑음}) * E(\text{경기}|\text{맑음}) + p(\text{흐림}) * E(\text{경기}|\text{흐림}) + p(\text{비}) * E(\text{경기}|\text{비})$$

$$E(\text{경기}|\text{날씨}) = \frac{5}{14}\left(-\frac{2}{5}log_2\left(\frac{2}{5}\right)-\frac{3}{5}log_2\left(\frac{3}{5}\right)\right) + \frac{4}{14}\left(-\frac{4}{4}log_2\left(\frac{4}{4}\right)-\frac{0}{4}log_2\left(\frac{0}{4}\right)\right) + \frac{5}{14}\left(-\frac{3}{5}log_2\left(\frac{3}{5}\right)-\frac{2}{5}log_2\left(\frac{2}{5}\right)\right) = 0.694$$

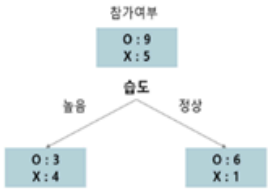


2-2) 온도

$$E(\text{경기}|\text{온도}) = \frac{4}{14}\left(-\frac{2}{4}log_2\left(\frac{2}{4}\right)-\frac{2}{4}log_2\left(\frac{2}{4}\right)\right) + \frac{6}{14}\left(-\frac{4}{6}log_2\left(\frac{4}{6}\right)-\frac{2}{6}log_2\left(\frac{2}{6}\right)\right) + \frac{4}{14}\left(-\frac{3}{4}log_2\left(\frac{3}{4}\right)-\frac{1}{4}log_2\left(\frac{1}{4}\right)\right) = 0.911$$

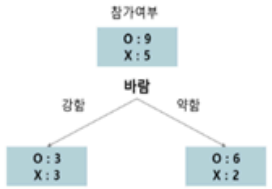
2-3) 습도

$$E(\text{경기}|\text{습도}) = \frac{7}{14}\left(-\frac{3}{7}log_2\left(\frac{3}{7}\right)-\frac{4}{7}log_2\left(\frac{4}{7}\right)\right) + \frac{7}{14}\left(-\frac{6}{7}log_2\left(\frac{6}{7}\right)-\frac{1}{7}log_2\left(\frac{1}{7}\right)\right) = 0.789$$



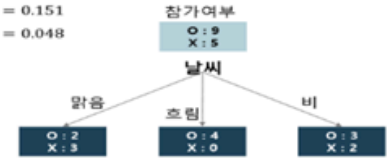
2-4) 바람

$$E(\text{경기}|\text{바람}) = \frac{6}{14}\left(-\frac{3}{6}log_2\left(\frac{3}{6}\right)-\frac{3}{6}log_2\left(\frac{3}{6}\right)\right) + \frac{8}{14}\left(-\frac{6}{8}log_2\left(\frac{6}{8}\right)-\frac{2}{8}log_2\left(\frac{2}{8}\right)\right) = 0.892$$



Approach 3) 각 속성에 대한 Information Gain 계산

$$IG(\text{경기, 날씨}) = E(\text{경기}) - E(\text{경기}|\text{날씨}) = 0.94 - 0.694 = 0.246$$ → 날씨로 가장 먼저 분류
$$IG(\text{경기, 온도}) = E(\text{경기}) - E(\text{경기}|\text{온도}) = 0.94 - 0.911 = 0.029$$
$$IG(\text{경기, 습도}) = E(\text{경기}) - E(\text{경기}|\text{습도}) = 0.94 - 0.789 = 0.151$$
$$IG(\text{경기, 바람}) = E(\text{경기}) - E(\text{경기}|\text{바람}) = 0.94 - 0.892 = 0.048$$



Approach 4) 각 terminal node에 대해 Approach 1, 2, 3 반복

예를 들어, "맑음"으로 분류된 노드는 "날씨 = 맑음" 인 데이터만 가지고 와서 다시 분할 전 엔트로피를 계산한다.

날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D11	맑음	포근	정상	강함	O

의사결정나무(Decision Tree) - CART 알고리즘, 지니계수

- CART 알고리즘은 불순도를 지니계수(Gini Index)로 계산
- 지니계수란?

  - 불순도를 측정하는 지표로서, 데이터의 통계적 분산정도를 정량화해서 표현한 값

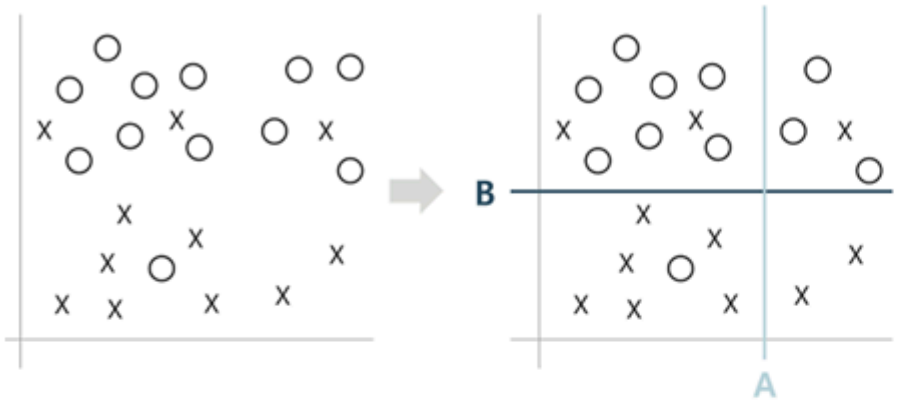
$$G(S) = 1 - \sum_{i=1}^{c} p_i^2$$

S : 이미 발생한 사건의 모음, c : 사건의 갯수

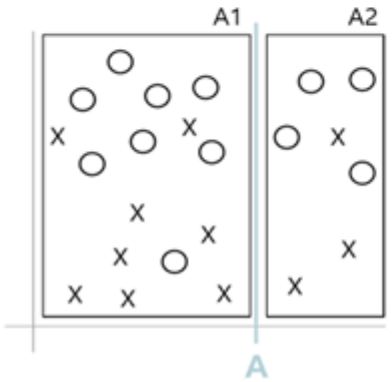
즉, Gini Index가 높을 수록 데이터가 분산되어있음을 의미한다.

CART 알고리즘

- A와 B중 O와 X를 더 잘 구분하는 기준은 무엇일까?



1) A를 기준으로 분할했을 때 지니계수

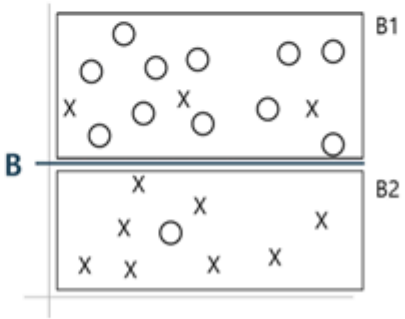


$$G_{A1} = 1 - \left(\frac{8}{16}\right)^2 - \left(\frac{8}{16}\right)^2 = 0.5$$

$$G_{A2} = 1 - \left(\frac{4}{7}\right)^2 - \left(\frac{3}{7}\right)^2 = 0.49$$

$$G_A = \left(\frac{16}{23}\right) * 0.5 + \left(\frac{7}{23}\right) * 0.49 = 0.497$$

2) B를 기준으로 분할했을 때 지니계수



$$G_{B1} = 1 - \left(\frac{11}{14}\right)^2 - \left(\frac{3}{14}\right)^2 = 0.34$$

$$G_{B2} = 1 - \left(\frac{1}{9}\right)^2 - \left(\frac{8}{9}\right)^2 = 0.2$$

$$G_B = \left(\frac{14}{23}\right) * 0.34 + \left(\frac{9}{23}\right) * 0.2 = 0.28$$

⛳DC 구현

```
#기본적인 의사결정나무의 형태
#sklearn 모듈의 tree import
from sklearn import tree

#간단한 데이터셋 생성
```

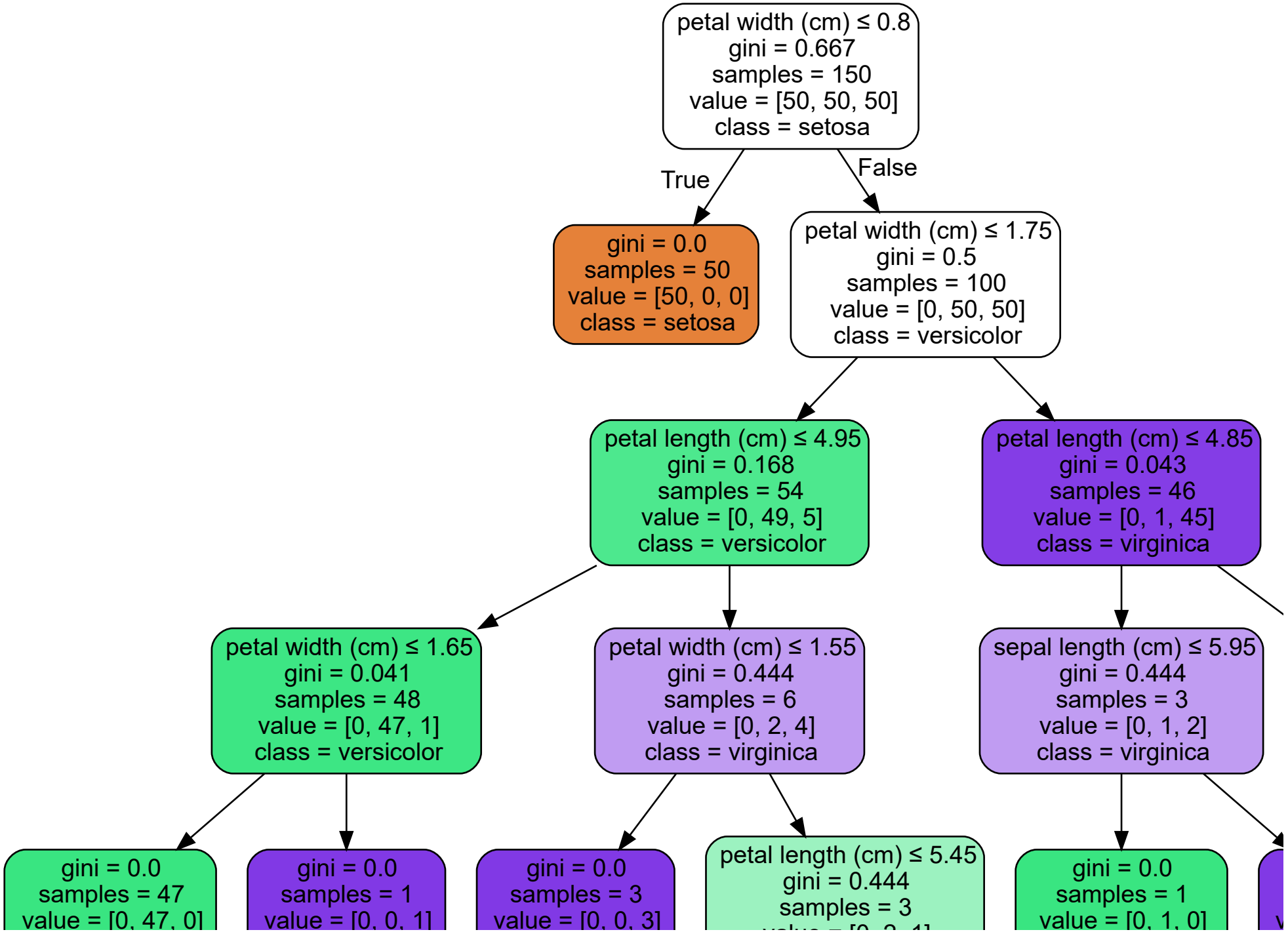
```
#학습데이터에 2개의 설명변수만 사용하여 2행짜리 데이터를 생성, 종속변수 또한 설명변수가 2행이기에 2개
X = [[0, 0], [1, 1]]
Y = [0, 1]

# 의사결정나무 적합 및 학습데이터 예측
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
clf.predict([[1, 1]])
```

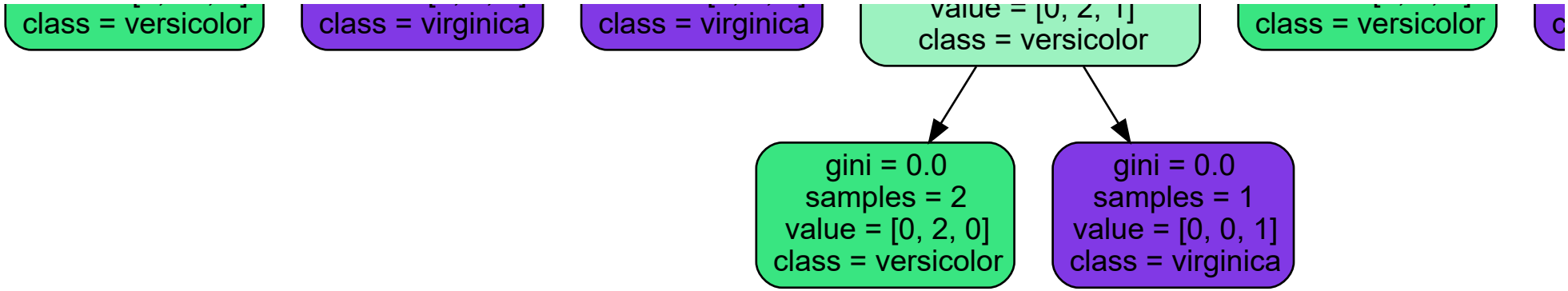
```
array([1])
```

```
#라이브러리 import & 실습 데이터 로드
#sklearn 모듈의 tree import
from sklearn import tree
from sklearn.datasets import load_iris
from os import system                # graphviz 라이브러리 설치를 위함

#graphviz 라이브러리 설치 // 아래 예제에서 오류나는 경우 anaconda prompt에서 설치바람
system("pip install graphviz")

# graphviz 사용에 있어서 error 발생원인이 환경변수일 경우 환경변수 추가 필요
# 환경변수 추가 후 환경변수 설정 아래코드
# os.environ["PATH"] += os.pathsep + 'C:WWProgram Files (x86)WWGraphviz2.38WWbinWW'

# iris 실습데이터 로드 - iris 데이터는 4개의 feature 변수가 있으며, 3개의 target 변수가 있음
iris = load_iris()
```

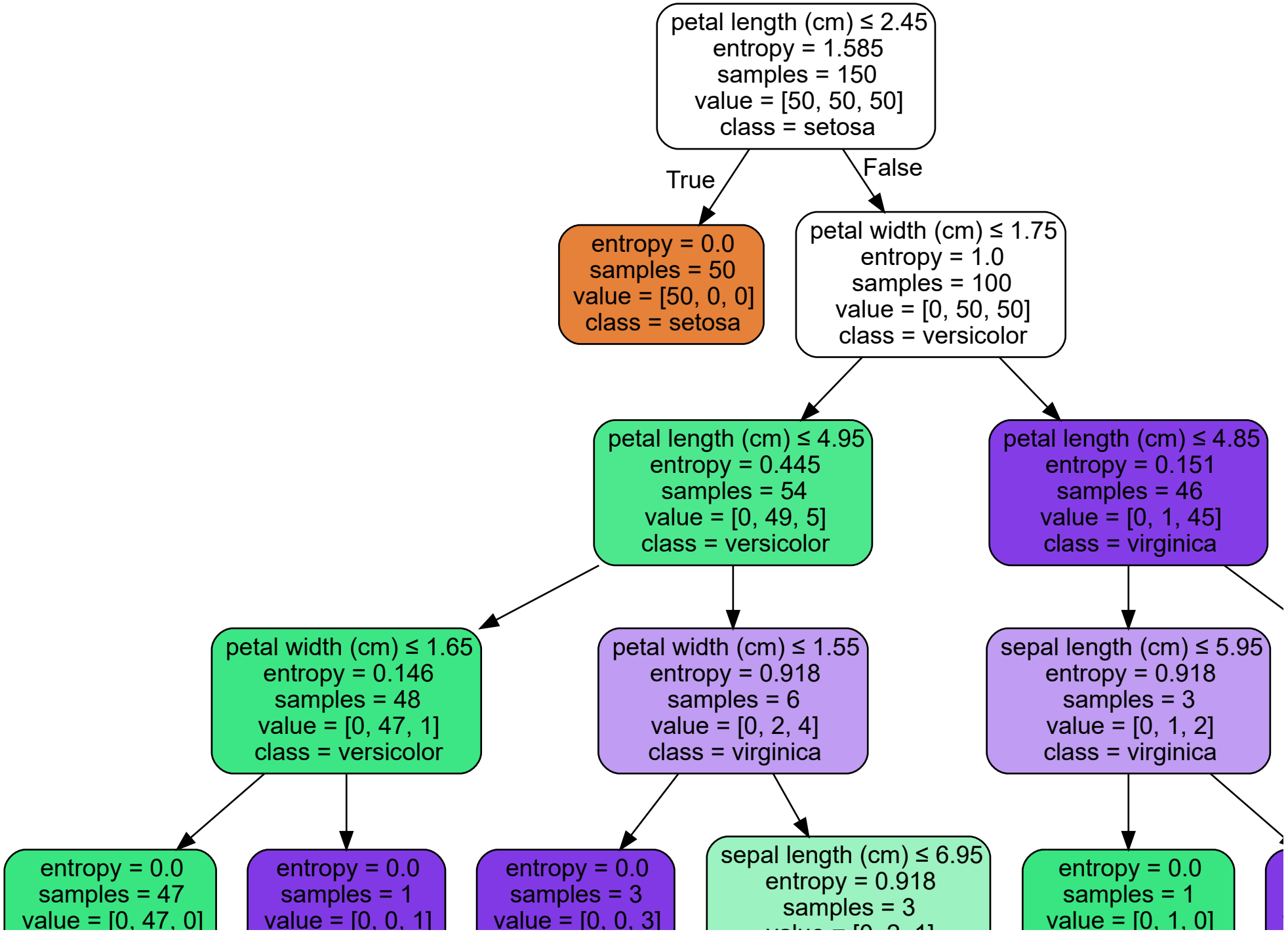
```
#Decision Tree Classifier(의사결정분류나무)
#기본적인 의사결정 나무 : Information Gain - Gini - Decision Tree를 이용할 때, 가장 기본적인(아무런 매개변수를 주지 않았을 때) Information Ga
#의사결정나무 분류
from sklearn.tree import export_graphviz
import graphviz

#gini계수는 엔트로피와 마찬가지로 낮을 수록 분류가 잘 된것으로 판단하며
#기본적으로 의사결정나무는 이 Information Gain을 낮추는 방향으로 분류를 진행

```

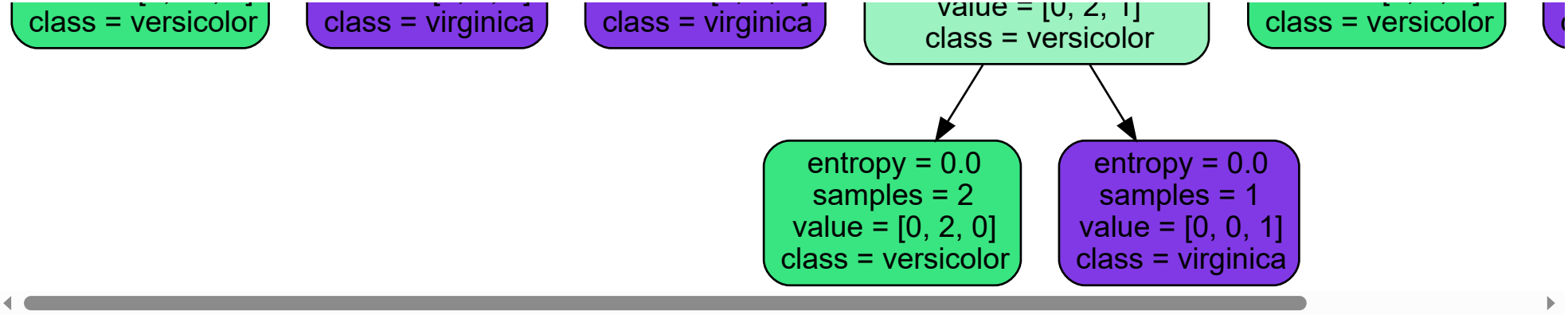
```
clf = tree.DecisionTreeClassifier()      #종속변수가 현재 범주형
clf = clf.fit(iris.data, iris.target)    #feature, target -  의사결정나무 분류모형을 적합시킨 것

# 시각화
dot_data = tree.export_graphviz(clf,   #의사결정나무 모형 대입
                                out_file = None,  #file로 변환할 것인가
                                feature_names = iris.feature_names,  #feature 이름
                                class_names = iris.target_names,  #target 이름
                                filled = True,           #그림에 색상을 넣을것인가
                                rounded = True,          #반올림을 진행할 것인가
                                special_characters = True)   #특수문자를 사용하나

graph = graphviz.Source(dot_data) #graphviz는 tree를 도식화하는 라이브러리
graph
```


petal width (cm) ≤ 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) ≤ 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) ≤ 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) ≤ 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) ≤ 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) ≤ 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal length (cm) ≤ 5.95
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
petal length (cm) ≤ 5.45
gini = 0.444
samples = 3
gini = 0.0
samples = 1
value = [0, 1, 0]




```
#Information Gain - entropy 의사결정나무
#의사결정나무 분류
#DecisionTreeClassifier()을 생성할 때 매개변수로 criterion = "entropy"만 추가
clf2 = tree.DecisionTreeClassifier(criterion = "entropy")  # Information Gain - entropy
clf2 = clf2.fit(iris.data, iris.target)                    # feature, target

#시각화
dot_data2 = tree.export_graphviz(clf2,        # 의사결정나무 모형 대입
                                 out_file = None,     # file로 변환할 것인가
                                 feature_names = iris.feature_names, # feature 이름
                                 class_names = iris.target_names,   # target 이름
                                 filled = True,        # 그림에 색상을 넣을것인가
                                 rounded = True,       # 반올림을 진행할 것인가
                                 special_characters = True)    # 특수문자를 사용하나

graph2 = graphviz.Source(dot_data2)
graph2
```


petal length (cm) ≤ 2.45
entropy = 1.585
samples = 150
value = [50, 50, 50]
class = setosa
True
False
entropy = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) ≤ 1.75
entropy = 1.0
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) ≤ 4.95
entropy = 0.445
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) ≤ 4.85
entropy = 0.151
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) ≤ 1.65
entropy = 0.146
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) ≤ 1.55
entropy = 0.918
samples = 6
value = [0, 2, 4]
class = virginica
sepal length (cm) ≤ 5.95
entropy = 0.918
samples = 3
value = [0, 1, 2]
class = virginica
entropy = 0.0
samples = 47
value = [0, 47, 0]
entropy = 0.0
samples = 1
value = [0, 0, 1]
entropy = 0.0
samples = 3
value = [0, 0, 3]
sepal length (cm) ≤ 6.95
entropy = 0.918
samples = 3
entropy = 0.0
samples = 1
value = [0, 1, 0]

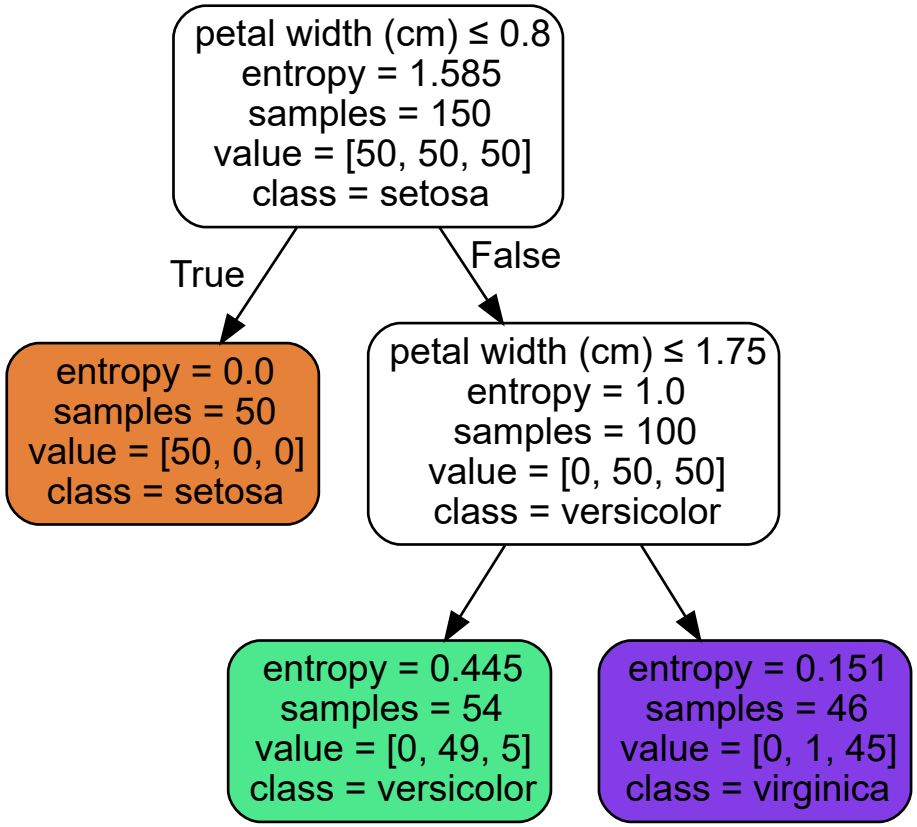



```
#두 의사결정나무 모형은 너무 많은 노드들이 존재
#마지막 노드에서 gini와 entropy 모두 0.0을 출력
#이는 완벽하게 분리시켰다고 말할 수 도 있지만, 사실 억지로 분류시킨 것에 가까움 - 그렇기에 과적합(Overfitting)이 발생!!!
#추가적으로 한 가지 더 알아야 할 것이 있음
#위의 색은 3가지 색의 계열로 이루어져 있음
#같은 색 계열이면 같은 집단으로 분류를 한 것이며, 색이 진할수록 Information Gain(entropy, gini .. )이 낮은 것임
#즉, 정확하게 분류를 했다는 것임
#이는 상대적이기에 depth가 작으면 entropy가 높아도 진하게 출력될 수 있음
```

```
#pruning(가지치기)라는 기법으로 과적합을 방지

clf3 = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = 2) #깊이제한을 2
clf3.fit(iris.data, iris.target)

#가지치기를 진행하는 방법은 여러 기준이 있는데....
#1) 지니계수/엔트로피와 같은 Information Gain의 값이 일정 수준 이하로 안내려가도록
#2) 가지의 개수 자체를 제한하는 방법
#3) 깊이를 제한하는 방법 등이 있음
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

```
#시각화
dot_data3 = tree.export_graphviz(clf3,                # 의사나무 모형 대입
                                 out_file = None,        # file로 변환할 것인가
                                 feature_names = iris.feature_names,  # feature 이름
                                 class_names = iris.target_names,   # target 이름
                                 filled = True,          # 그림에 색상을 넣을것인가
                                 rounded = True,         # 반올림을 진행할 것인가
                                 special_characters = True)  # 특수문자를 사용하나

graph3 = graphviz.Source(dot_data3)
graph3
#가지치기의 기준으로 max_depth를 2로 주었더니, 트리의 깊이가 2로 변함
#entropy 또한 0.4/0.151로 많이 높아졌음
#위의 DecisionTree의 gini/entropy는 0.00이었는데 분류가 너무 안된 것이 아닌가? 라고 생각할 수 있음
#train 데이터를 예측했기에 학습데이터의 경우 가지가 무한정 많아지면 정확해질 수 밖에 없dma
#만약 새로운 test 데이터가 주어진다면, 오히려 과적합된 DecisionTree가 학습데이터 내에서
#너무 이상값들에 집중해서 일반적인 새로운 test 데이터를 제대로 예측하지 못할 수도 있음
#게다가 이 짧은 트리가 거창한 트리보다 훨씬 직관적이고 이해도가 높음
#Decision Tree를 사용하는 가장 큰 이유중 하나가 바로 "직관적인 이해"인데 가지치기를 하지않고
#무한한 가지를 만들면 Decision Tree를 사용하는 의미 또한 퇴색됨
```




```
#Confusion matrix를 활용하여 3가지 분류기의 학습데이터를 분류하는 정확도를 확인
from sklearn.metrics import confusion_matrix

# 1번 의사결정나무 - 지니계수 활용
confusion_matrix(iris.target, clf.predict(iris.data))
```

```
array([[50,  0,  0],
       [ 0, 50,  0],
       [ 0,  0, 50]])
```

```
# 2번 의사결정나무 - entropy 활용
confusion_matrix(iris.target, clf2.predict(iris.data))
```

```
array([[50,  0,  0],
       [ 0, 50,  0],
       [ 0,  0, 50]])
```

```
# 3번 의사결정나무 - 가지치기 작업
confusion_matrix(iris.target, clf3.predict(iris.data))
#가지치기를 한 의사결정나무의 정확도가 가장 떨어wla
#하지만 학습데이터를 분류한 것이라는 사실을 염두해 두어야 gka
#만약 새로운 데이터가 들어오면 말했듯이 맨 마지막 가지치기의 의사결정나무가
#일반화된 특징을 잡을 가능성이 높음
```

```
array([[50,  0,  0],
       [ 0, 49,  1],
       [ 0,  5, 45]])
```

```
#Traing Set / Test Set 구분
#지금까지 전부 학습데이터로 분류를 진행했지만 실제 데이터가 주어졌을 때,
#데이터는 Train/(Validation)/Test로 나누어 학습할 가능성이 큼
#그렇기에 Train set과 Test set을 나누어 실습
# 데이터셋 분리 함수
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris.data     #feature
                                                    , iris.target     #target
                                                    , stratify = iris.target #층화추출법
                                                    , random_state = 1)  #난수고정
#train_test_split의 매개변수중 stratify 매개변수가 들어감
#이것은 필수적으로 들어가야할 요소는 아님
#iris데이터셋의 경우 150개의 데이터밖에 없기에 무작위 추출이 진행된다면 target 데이터가 치우쳐질 수도 있음
#이러한 사용은 제약/임상실험에서도 마찬가지임(병에 안걸린사람이 걸린사람보다 훨씬 많으니 과소평가될 가능성이 있음)
#그렇기에 데이터가 적은 이유로 고루고루 데이터를 추출시키기 위해 층화추출법을 사용한 것임
```

```
# train dataset
clf4 = tree.DecisionTreeClassifier(criterion = "entropy")
clf4.fit(X_train, y_train)
```

```
4
5 # test set predict confusion matrix
6 confusion_matrix(y_test,clf4.predict(X_test))
7 #confusion matrix결과를 보면 가지치기를 하지 않았는데도 불구하고
8 #3번에서 한 경우에 오분류가 발생함
9 #이는 train data set과 test data set의 특성이 어느정도 달라서 학습의 분류결과가
10 #완전하게 맞을 순 없다는 것을 보여줌
```

```
array([[12,  0,  0],
       [ 0, 13,  0],
       [ 0,  1, 12]])
```

```
1 #Decision regression Tree(의사결정회귀나무)
2 #의사결정 회귀나무는 종속변수가 연속형 변수일때 진행
3 #기본적인 방식은 의사결정 분류나무와 동일하나 사용하는 함수가 다름
4 # 필요 라이브러리
5 import numpy as np
6 from sklearn.tree import DecisionTreeRegressor    # 회귀나무 함수
7 import matplotlib.pyplot as plt
8
9 # 실습용 데이터셋 만들기
10 rng = np.random.RandomState(1)
11 X = np.sort(5 * rng.rand(80, 1), axis=0)
12 y = np.sin(X).ravel()                          # sin함수의 예측을 목표로한다
13 y[::5] += 3 * (0.5 - rng.rand(16))             # 이상치를 발생시킨다.
```

```
1 # X_test set 생성
2 X_test = np.arange(0.0,5.0,0.01)[:,np.newaxis]
```

```
1 #Regression  Tree 구축
2 # 깊이가 다른 두 Regression 나무 생성
3 regr1 = tree.DecisionTreeRegressor(max_depth = 2)
4 regr2 = tree.DecisionTreeRegressor(max_depth = 5)
```

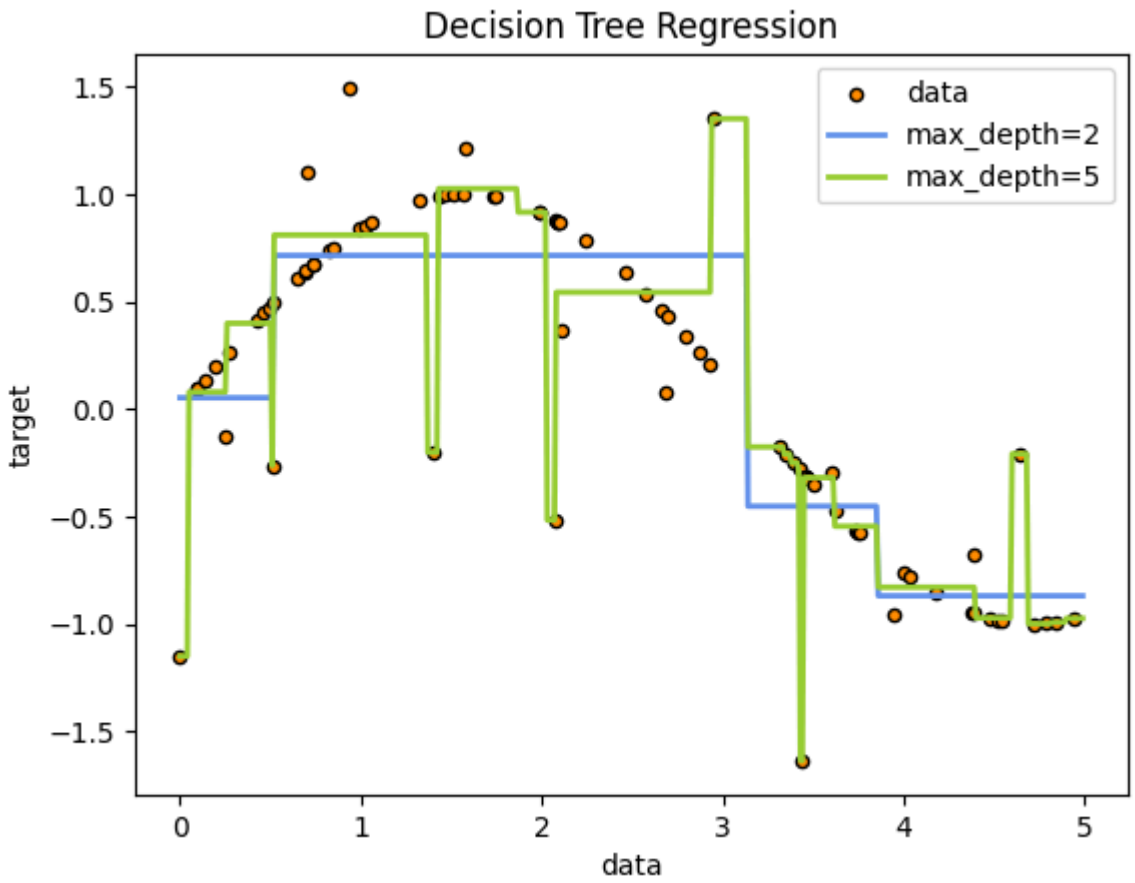
```python

# 두 가지 회귀나무 적합
regr1.fit(X,y)
regr2.fit(X,y)
# 예측
y_1 = regr1.predict(X_test)
y_2 = regr2.predict(X_test)
# 예측 결과물
y_1
```

```
array([ 0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.05236068,  0.05236068,  0.05236068,
        0.05236068,  0.05236068,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
        0.71382568,  0.71382568,  0.71382568,  0.71382568,  0.71382568,
```

```
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568, 0.71382568, 0.71382568, 0.71382568, 0.71382568,
       0.71382568  0.71382568  0.71382568  0.71382568  0.71382568
```

```
# depth가 다른 두 회귀나무 도식화

plt.figure()
plt.scatter(X, y, s=20, edgecolor="black",
            c="darkorange", label="data")
plt.plot(X_test, y_1, color="cornflowerblue",
         label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="yellowgreen", label="max_depth=5", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
```

```
plt.legend()
plt.show()
#max_depth = 5인 의사결정회귀나무는 이상값에 영향을 더 크게 받았음을 확인할 수 있음
#오히려 max_depth = 2의 의사결정회귀나무가 이상값을 무시하고 전체적인 추세를 더 잘 잡는 것을 확인할 수 있음
#만약 sin함수에서 떨어져있는 점들이 이상값이 아니었다면, 저런 점들 또한 고려할 필요가 생김
#그렇기에 가지치기의 적절한 기준을 찾는 것 또한 분석가의 안목에 달려있음
#회귀트리는 각 데이터들이 유사한 y값으로 모여있는 부분으로 구역을 나눈 뒤 해당 구역에서 y값들의 평균값을 x축에 평행한 선을 그어 나타냈고,
#이 선들을 이어서 계단형태의 분류트리를 만들어 냄
#https://jaylala.tistory.com/entry/머신러닝-with-Python-회귀-트리Decision-Tree?category=1211276 [Innov_AI_te:티스토리]
```




```
# depth = 5 의사결정 회귀나무 시각화

```

```
dot_data4 = tree.export_graphviz(regr2, out_file=None,
                                 filled=True, rounded=True,
                                 special_characters=True)
graph4 = graphviz.Source(dot_data4)
graph4
#depth가 5인 의사결정나무의 부분임
#이미지가 너무 커서 짤렸으나 한 가지 확인하고 가야할 것이 있음
#위에서 보면 value가 낮을 수록 같은 색계열에서 연한 색을 띔
#그리고 value값이 높을 수록 진한 색을 띔
#https://todayisbetterthanyesterday.tistory.com/38
```