머신러닝 – 분류의 개요와 결정트리

BITAmin 9주차 정규 Session

1조 – 김범준, 박소영, 오세정, 이현진

목차

1. 분류의 개요

분류의 개요

분 류

- 학습 데이터로 주어진 데이터의 피처와 레이블값(결정 값, 클래스 값)을 머신러닝 알고리즘으로 학습해 모델을 생성, 이렇게 생성된 모델에 새로운 데이터 값이 주어졌을 때 미지의 레이블 값을 예측
- 기존 데이터가 어떤 레이블에 속하는지 패턴을 알고리즘으로 인지한 뒤에 새롭게 관측된 데이터에 대한 레이블 판별
- 지도학습의 대표적인 예시

→ **명시적인 정답이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식**

분류의 개요

분류 방식의 머신러닝 알고리즘

- 베이즈(Bayes) 통계와 생성 모델에 기반한 나이브 베이즈(Naïve Bayes)
- 독립변수와 종속변수의 선형 관계성에 기반한 로지스틱 회귀(Logistic Regression)
- 데이터 균일도에 따른 규칙 기반의 결정 트리(Decision Tree)
- 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 서포트 벡터 머신(Support Vector Machine)
- 근접 거리를 기준으로 하는 최소 근접(Nearest Neighbor) 알고리즘
- 심층 연결 기반의 신경망 (Neural Network)
- 서로 다른(또는 같은) 머신러닝 알고리즘을 결합한 앙상블(Ensemble)

분류의 개요

앙상블 기법 : 정형 데이터 예측 분석 영역에서 매우 높은 예측 성능으로 인해 많은 분석가와 데이터 과학자에게 애용되는 방법 → 서로 다른/또는 같은 알고리즘을 결합하는 방법

(다음주 내용입니다)

앙상블의 기본 알고리즘으로 사용되는 것이
오늘 배울 결정 트리 기법

분류의 개요

결정 트리

장점

매우 쉽고 유연하게 적용될 수 있는 알고리즘
데이터의 스케일링이나 정규화 등 사전 가공의 영향 少

단점

예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 하는데
이로 인해 과적합(overfitting)이 발생해 예측 성능이 저하될 수 있음

분류의 개요

결정 트리

장점

데이터의 스케일링이나 정규화 등 사전 가공의 영향 少

단점

예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 하는데
이로 인해 과적합(overfitting)이 발생해 예측 성능이 저하될 수 있음

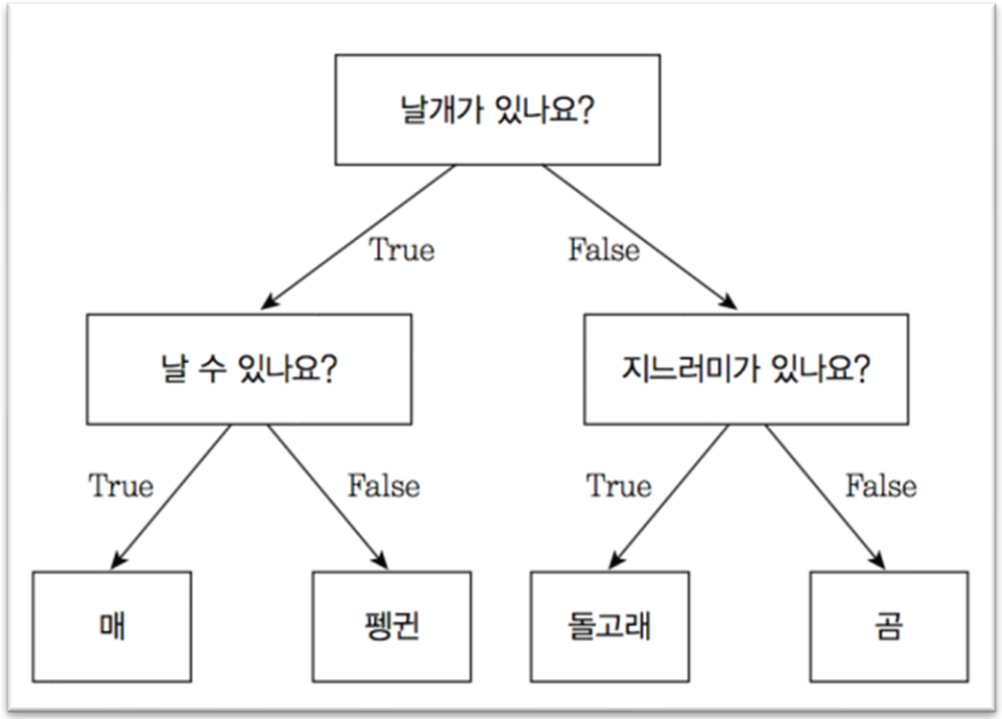
앙상블 기법에서는 이러한 단점이 오히려 장점으로 작용

앙상블은 매우 많은 여러 개의 약한 학습기(=예측 성능이 떨어지는 알고리즘)를
결합해 확률적 보완과 오류가 발생한 부분에 대한 가중치를 계속해서
업데이트하면서 예측 성능을 향상시키는데 이 때 결정 트리가
좋은 ' 약한 학습기 ' 가 되어주기 때문

2. 결정 트리 모델의 특징

결정 트리의 개념

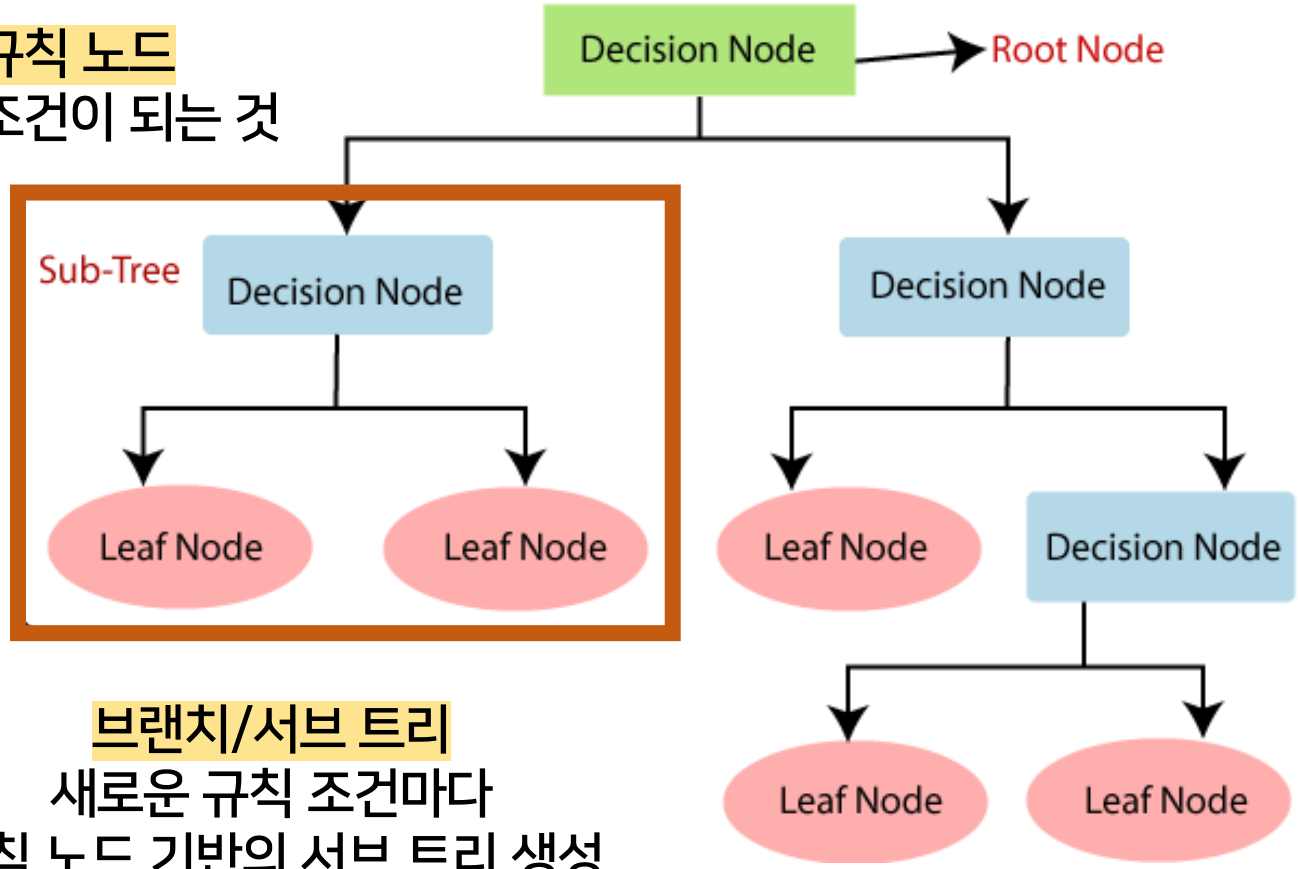
- 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리(Tree) 기반의 분류 규칙을 만드는 것 (if / else)
- 데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인지가 알고리즘의 성능을 크게 좌우
- 분류와 회귀 모두 사용 가능



결정 트리의 개념

규칙 노드
규칙 조건이 되는 것

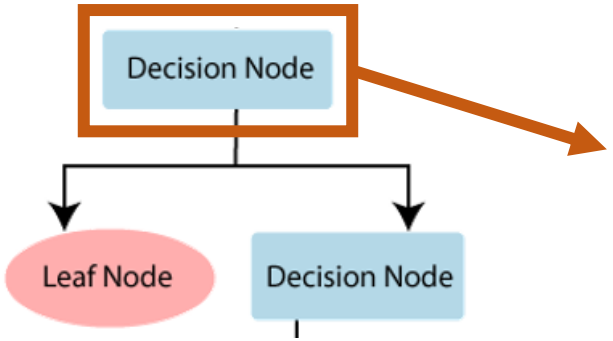
리프 노드
결정된 클래스 값



브랜치/서브 트리
새로운 규칙 조건마다
규칙 노드 기반의 서브 트리 생성

결정트리 모델의 특징

- 데이터 세트에 피처가 있고 이러한 피처가 결합해 규칙 조건을 만들 때마다 규칙 노드 생성



규칙 노드가 단일 특성의 데이터를 평가

- 많은 규칙이 있다는 것은 곧 분류를 결정하는 방식이 복잡해진다는 것이며 과적합으로 이어지기 쉬움

  → 트리의 깊이(depth)가 깊어질수록 결정 트리의 예측 성능 저하 가능성 大

- 가능한 한 적은 결정 노드로 높은 예측 정확도를 가지려면 데이터를 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 함

결정트리 모델의 특징

- 데이터 세트에 피처가 있고 이러한 피처가 결합해 규칙 조건을 만들 때마다 규칙 노드 생성

규칙 노드가 단일 특성의
데이터를 평가

이를 위해서 어떻게 트리를 분할(Split)해야 할까?
최대한 균일한 데이터 세트를 구성하도록 분할하는 게 좋음

- 많은 규칙이 있다는 것은 곧 분류를 결정하는 방식이 복잡해진다는 것이며 과적합으로 이어지기 쉬움

  → 트리의 깊이(depth)가 깊어질수록 결정 트리의 예측 성능 저하 가능성 大

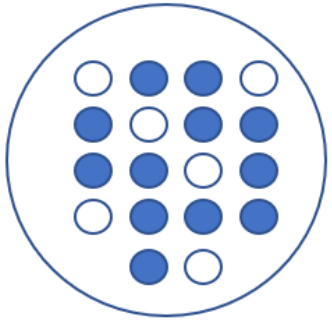
- 가능한 한 적은 결정 노드로 높은 예측 정확도를 가지려면 데이터를 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 함

결정 트리 모델의 특징

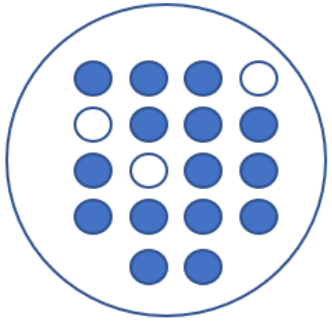
- 균일한 데이터 세트부터 순서대로 나열한다면?

상대적으로
혼잡도가 높고
균일도가 낮아
더 많은 정보 필요

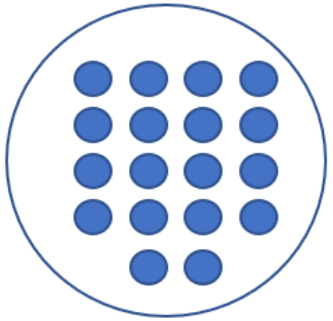
A



B



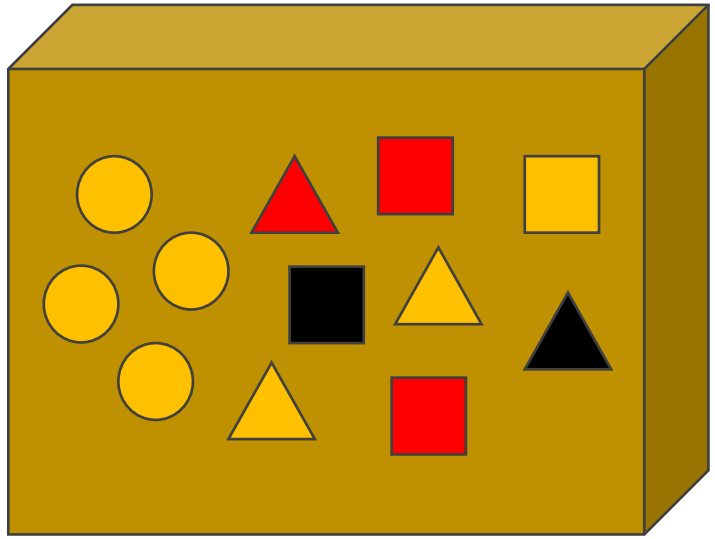
C



데이터에 별다른
정보가 없어도
파란 공이라고
쉽게 예측 가능

C → B → A 순으로 균일도가 높음

결정 트리 모델의 특징



각 레고 블록을 형태와 색깔 속성으로 분류

가장 첫 번째로 만들어져야 하는 규칙은
If 색깔 == '동그라미'
동그라미 블록이면 모두 노란색 블록이기 때문에
가장 쉽게 예측할 수 있기 때문

정보의 균일도를 측정하는 대표적인 방법으로
엔트로피를 이용한 **정보이득지수**와 **지니계수**가 있음

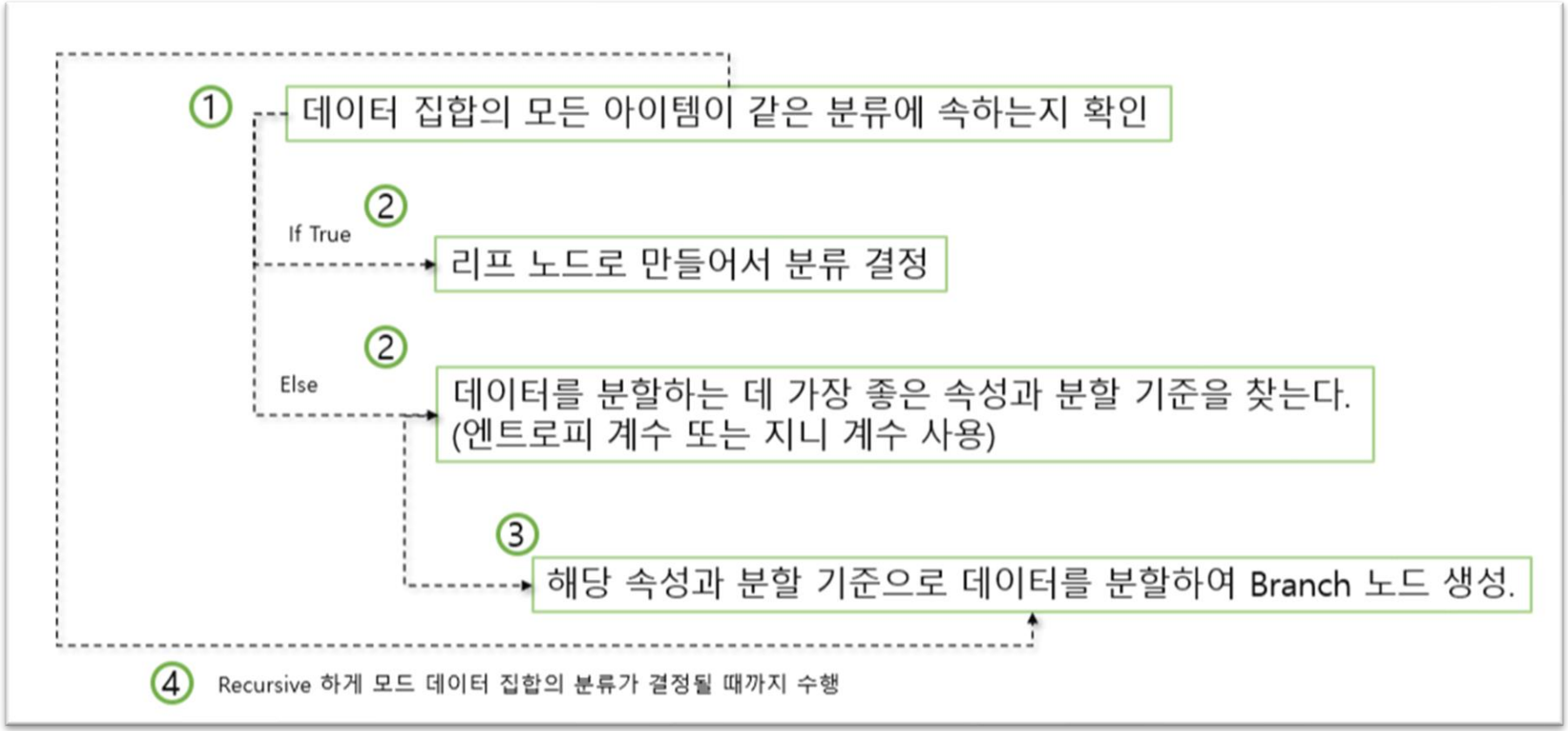
정보의 균일도 측정

- 엔트로피: 엔트로피는 주어진 데이터 집합의 혼잡도를 의미하는데, 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮음. 정보 이득 지수 = (1-엔트로피 지수)이며 결정 트리는 정보 이득이 높은 속성을 기준으로 분할

- 지니계수: 경제학의 불평등 지수에서 따왔으며 0이 가장 평등하고 1로 갈수록 불평등함. 머신러닝에 적용될 때는 지니 계수가 낮을수록 데이터 균일도가 높은 것으로 해석해 지니 계수가 낮은 속성을 기준으로 분할

순도(homogeneity)를 높이고 불순도(불확실성, uncertainty) 감소

정보의 균일도 측정

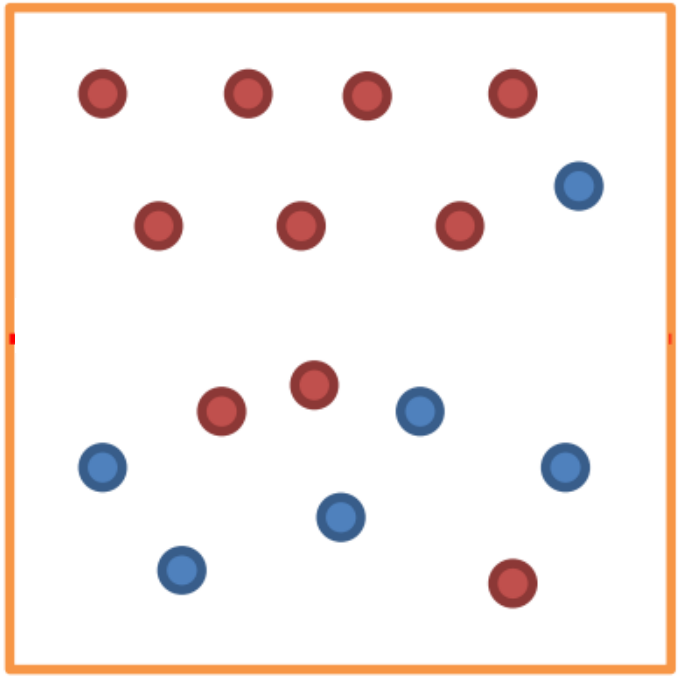

① 데이터 집합의 모든 아이템이 같은 분류에 속하는지 확인
② If True
리프 노드로 만들어서 분류 결정
② Else
데이터를 분할하는 데 가장 좋은 속성과 분할 기준을 찾는다.
(엔트로피 계수 또는 지니 계수 사용)
③ 해당 속성과 분할 기준으로 데이터를 분할하여 Branch 노드 생성.
④ Recursive 하게 모드 데이터 집합의 분류가 결정될 때까지 수행

정보의 균일도 측정 (엔트로피)

엔트로피 도출 식: $Entropy(A) = -\sum_{k=1}^{m} p_k \log_2 (p_k)$

전체 16개 가운데 빨간색 동그라미는 10개, 파란색 동그라미는 6개
이를 바탕으로 A 영역의 엔트로피 도출

$$Entropy(A) = -\frac{10}{16}\log_2 (\frac{10}{16}) - \frac{6}{16}\log_2 (\frac{6}{16}) \approx 0.95$$



정보의 균일도 측정 (엔트로피)

두 개 이상 영역에 대한 엔트로피 공식:

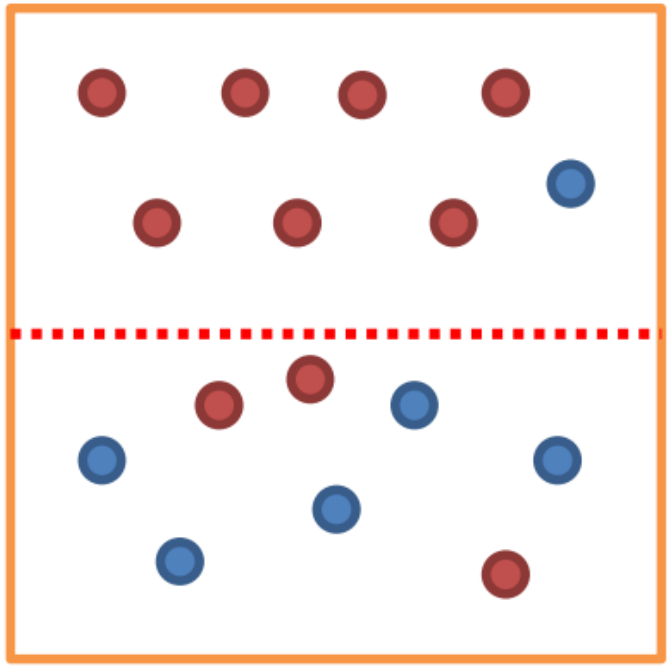
$$Entropy(A) = \sum_{i=1}^{d} R_i \left(- \sum_{k=1}^{m} p_k \log_2 (p_k) \right)$$

이를 바탕으로 새로운 엔트로피 도출

$$Entropy(A) = 0.5 \times \left(-\frac{7}{8}\log_2 (\frac{7}{8}) - \frac{1}{8}\log_2 (\frac{1}{8}) \right) + 0.5 \times \left(-\frac{3}{8}\log_2 (\frac{3}{8}) - \frac{5}{8}\log_2 (\frac{5}{8}) \right) \approx 0.75$$

분할 이후 엔트로피가 0.2 감소 → 불순도 감소

박스를 두 개로 분할



정보의 균일도 측정 (엔트로피)

두 개 이상 영역에 대한 엔트로피 공식:

$$Entropy(A) = \sum_{i=1}^{d} R_i \left(-\sum_{k=1}^{m} p_k \log_2 (p_k) \right)$$

이를 바탕으로 새로운 엔트로피 모델

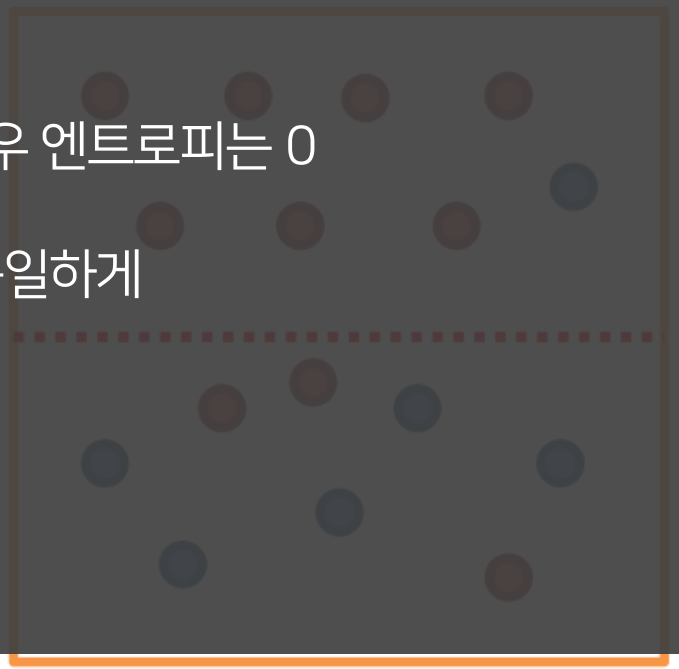
A 영역에 속한 모든 레코드가 동일한 범주에 속할 경우 엔트로피는 0

반대로 범주가 둘뿐이고 해당 개체의 수가 동일하게
반반씩 섞여 있을 경우 엔트로피는 1

$$Entropy(A) = 0.5 \times \left(-\frac{7}{8}\log_2 \left(\frac{7}{8}\right) - \frac{1}{8}\log_2 \left(\frac{1}{8}\right) \right) + 0.5 \times \left(-\frac{3}{8}\log_2 \left(\frac{3}{8}\right) - \frac{5}{8}\log_2 \left(\frac{5}{8}\right) \right) \approx 0.75$$

분할 이후 엔트로피가 0.2 감소 → 불순도 감소

박스를 두 개로 분할



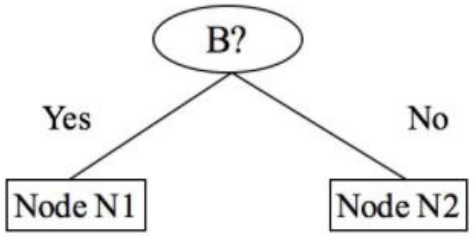
정보의 균일도 측정 (지니계수)

Data set T: n개의 class label
p_j는 T 안에서의 class j의 상대 빈도

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

T가 2개의 T1, T2로 분할되고,
각각 데이터의 개수가 N1, N2개 일 때

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$



	Parent
C1	7
C2	5

Gini(N1)
= 1 – (5/6)² – (1/6)²
= 0.278

Gini(N2)
= 1 – (2/6)² – (4/6)²
= 0.444

	N1	N2
C1	5	2
C2	1	4
Gini=0.361		

Weighted Gini of N1 N2
= 6/12 * 0.278 +
6/12 * 0.444
= 0.361

결정 트리 모델의 특징

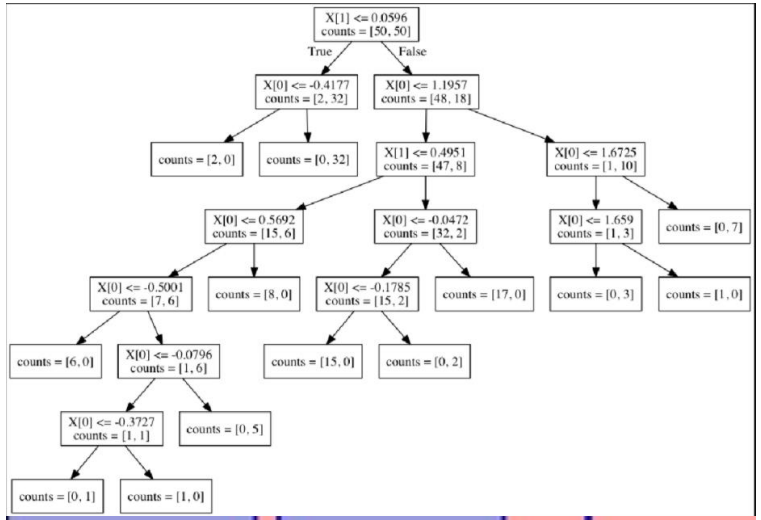
- 장점

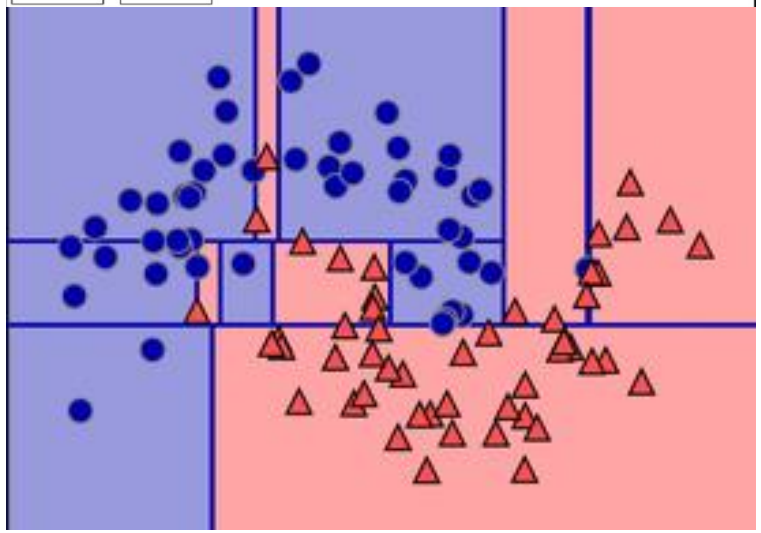
- 균일도 기반, 쉽고 직관적
- 룰 명확, 시각화 표현 가능
- 사전 가공(정규화 등) 영향도 크지 X

- 단점

- 과적합 문제
- 균일도 다양할 수록 트리 복잡

→ 복잡한 모델은 test 세트에 유연하게 대처X



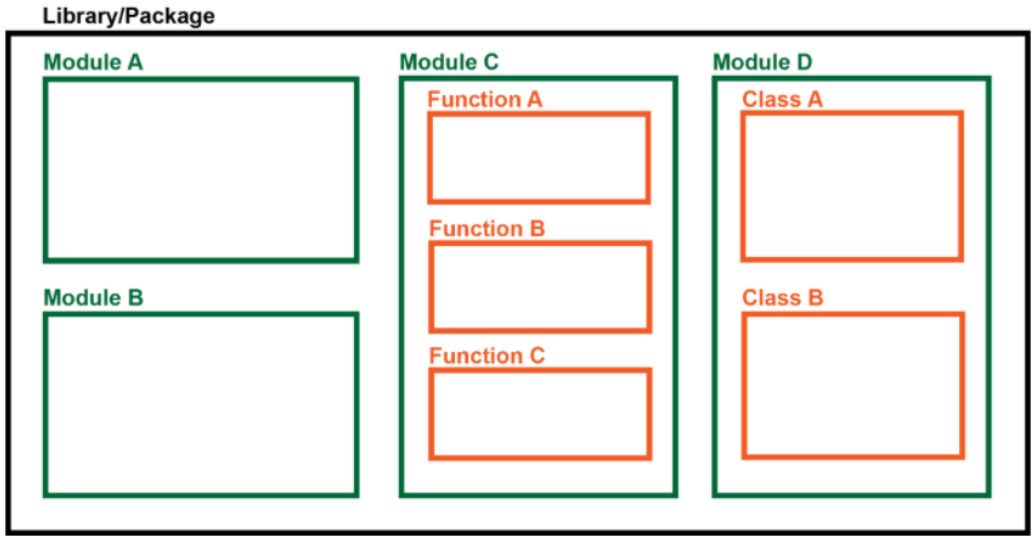


3. 결정 트리 파라미터

싸이킷런

- 싸이킷런 (Scikit-learn)이란?

파이썬 머신러닝 라이브러리 중 가장 많이 사용되는 라이브러리
설치 방법: Anaconda prompt 또는 Jupyter에서 'pip install sklearn' 입력



싸이킷런

싸이킷런 주요 모듈

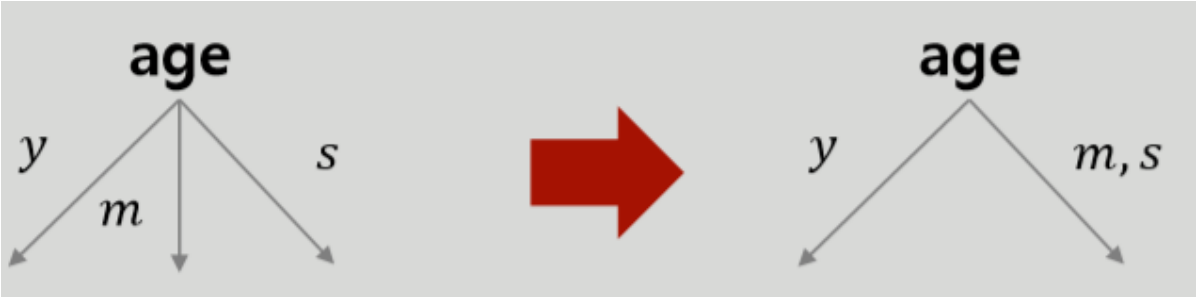
분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업을 수행하는 다양한 기능 제공
	sklearn.feature_extraction	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 Tf-Idf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 sklearn.feature_extraction.text 모듈에, 이미지 데이터의 피처 추출은 sklearn.feature_extraction.image 모듈에 지원 API가 있음
피처 처리 & 차원 축소	sklearn.decomposition	차원 축소와 관련한 알고리즘을 지원하는 모듈이다. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있다.
데이터 분리, 검증 & 파라미터 튜닝	sklearn.model_selection	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	sklearn.metrics	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	sklearn.ensemble	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공
	sklearn.linear_model	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Desccent) 관련 알고리즘도 제공
	sklearn.naive_bayes	나이브 베이즈 알고리즘 제공. 가우시안 NB. 다항 분포 NB 등
	sklearn.neighbors	최근접 이웃 알고리즘 제공. K-NN(K-Nearest Neighborhood) 등
	sklearn.svm	서포트 벡터 머신 알고리즘 제공
	sklearn.tree	의사 결정 트리 알고리즘 제공
	sklearn.cluster	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
유틸리티	sklearn.pipeline	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

sklearn.tree

- Class

분류: DecisionTreeClassifier

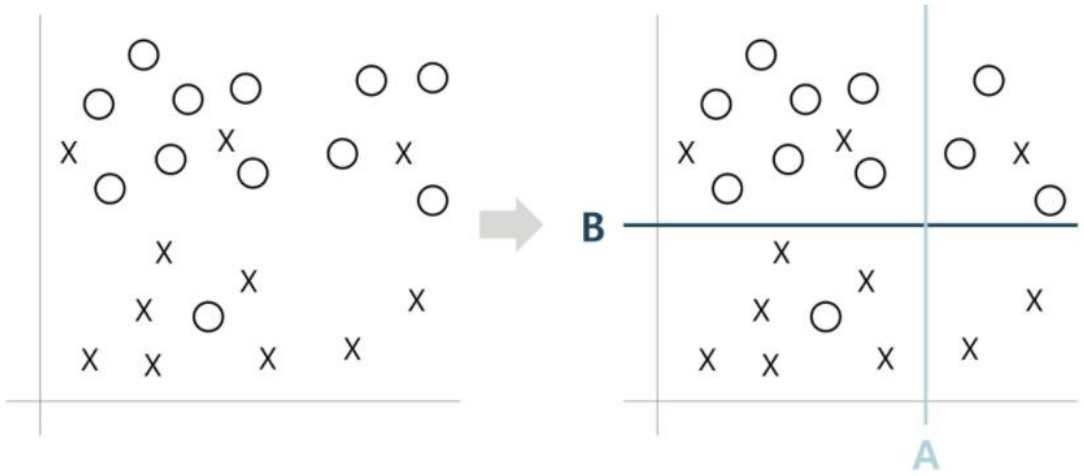
회귀: DecisionTreeRegressor



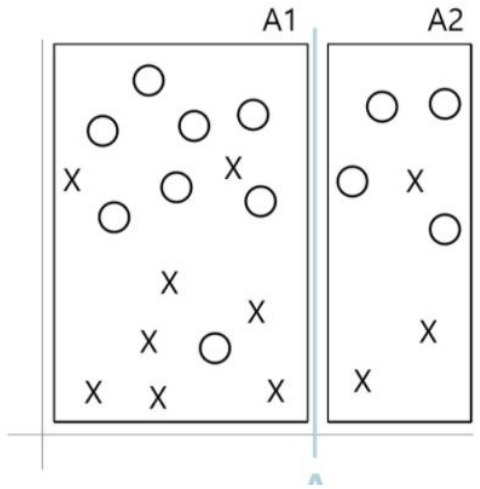
- CART(Classification And Regression Trees) 알고리즘 기반
  - 의사결정 트리 방법론 중 하나
  - 지니 지수(불확실성) 또는 분산의 감소량을 이용하여 binary split 수행

CART 알고리즘의 예시

A와 B 중 어떤 기준으로 먼저 분할?



1) A를 기준으로 분할했을 때 지니계수

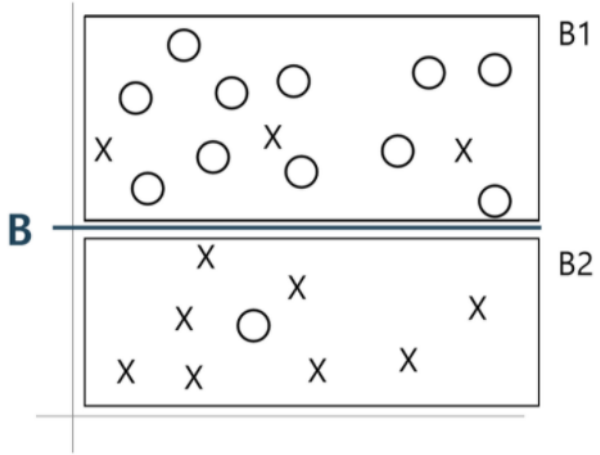


$$G_{A1} = 1 - \left(\frac{8}{16}\right)^2 - \left(\frac{8}{16}\right)^2 = 0.5$$

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

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

2) B를 기준으로 분할했을 때 지니계수



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

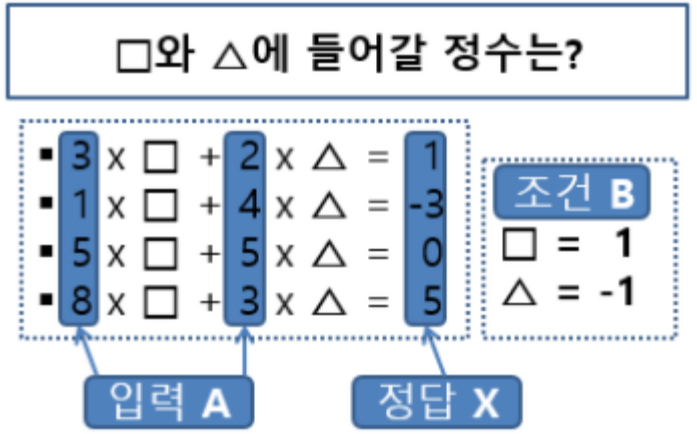
$$G_{B2} = 1 - \left(\frac{1}{9}\right)^2 - \left(\frac{8}{9}\right)^2 = 0.2$$

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

결정 트리 파라미터

Parameter란?

- 모델 내부에서 자동으로 결정되는 변수
- 학습 후) 데이터 분석을 통해 얻은 값



Hyper Parameter란?

- 학습 전) 모델을 생성할 때, 사용자가 직접 설정하는 변수
- 편의 상 파라미터라고 표현함

→ "모델의 (하이퍼) 파라미터를 조정한다"

결정 트리 파라미터

DecisionTreeClassifier 파라미터

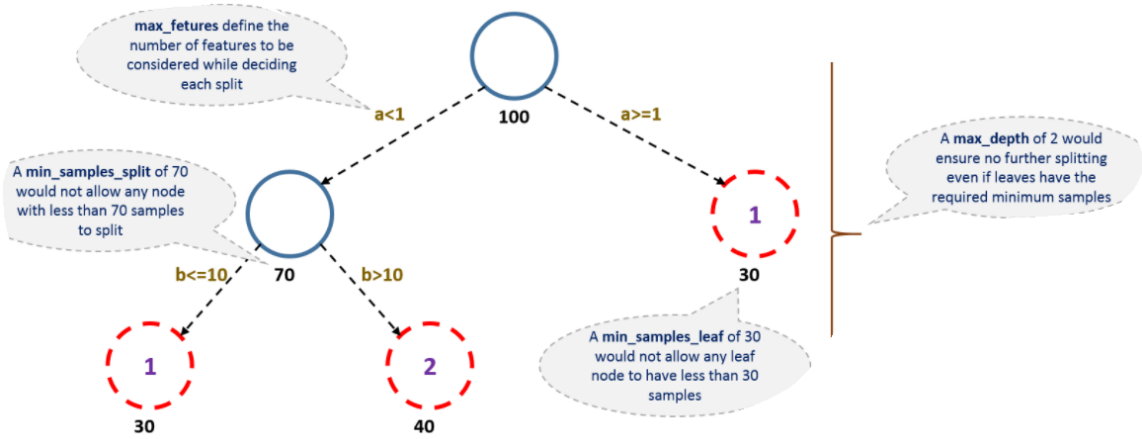
- criterion : 분할 품질을 측정하는 기능 (default : gini)
- splitter : 각 노드에서 분할을 선택하는 데 사용되는 전략 (default : best)
- max_depth : 트리의 최대 깊이 (값이 클수록 모델의 복잡도가 올라간다.)
- min_samples_split : 자식 노드를 분할하는데 필요한 최소 샘플 수 (default : 2)
- min_samples_leaf : 리프 노드에 있어야 할 최소 샘플 수 (default : 1)
- min_weight_fraction_leaf : min_sample_leaf와 같지만 가중치가 부여된 샘플 수에서의 비율
- max_features : 각 노드에서 분할에 사용할 특징의 최대 수
- random_state : 난수 seed 설정
- max_leaf_nodes : 리프 노드의 최대수
- min_impurity_decrease : 최소 불순도
- min_impurity_split : 나무 성장을 멈추기 위한 임계치
- class_weight : 클래스 가중치
- presort : 데이터 정렬 필요 여부

결정 트리 파라미터

DecisionTreeClassifier 파라미터

DecisionTreeClassifier(criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, random_state, max_leaf_nodes, min_impurity_decrease, min_impurity_split, class_weight, presort)

파라미터명	설명
min_samples_split	노드를 분할하기 위한 최소한의 샘플 데이터 수
min_samples_leaf	말단 노드(leaf)가 되귀 위한 최소한의 샘플 데이터 수
max_feautres	최적의 분할을 위해 고려할 최대 피처 개수
max_depth	트리의 최대 깊이를 규정
max_leaf_nodes	말단 노드의 최대 개수



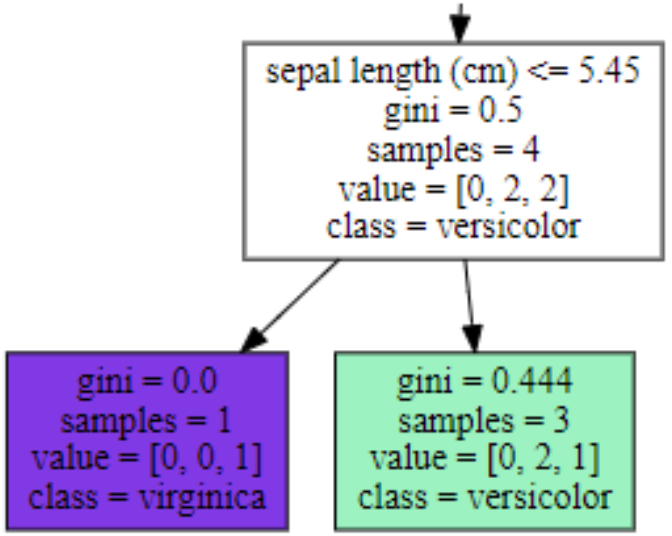
결정 트리 파라미터

min_samples_split

- 노드에서 가지 분리할 때 필요한 최소 sample 개수 (default=2)
- 작게 설정 → 분할되는 노드↑ → 과적합↑
- 원하는 값을 설정함으로써 과적합 제어 가능

ex) min_samples_split = 4로 설정
-> sample의 개수가 4개 이상일 경우에만 분리함

클수록 과적합 방지
작을수록 과적합 위험



결정 트리 파라미터

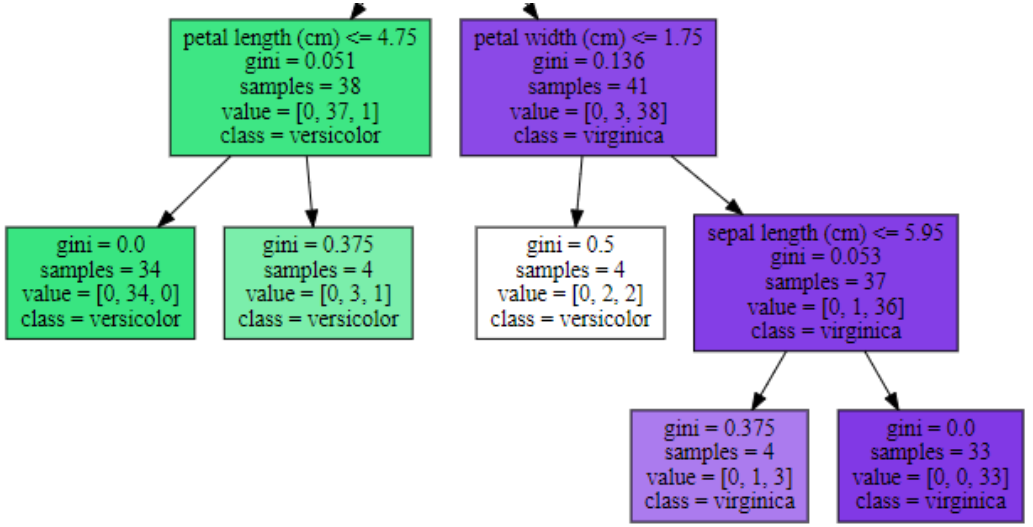
min_samples_leaf

- 말단 노드(leaf)가 되기 위한 최소한의 sample 데이터 수
- 과적합 제어 용도

* 주의 : 비대칭적 데이터 경우, 특정 클래스 데이터가 매우 작을 수 있음! → 작게 설정

ex) min_samples_leaf = 4로 설정
-> 말단 leaf sample 개수가 4개 이상

클수록 과적합 방지
작을수록 과적합 위험



결정 트리 파라미터

max_features

최적의 분할을 위해 고려할 최대 피처 개수

- default='None': 모든 피처를 사용해 분할
- Int 형 지정: 대상 피처의 개수 (ex. max_features = 2)
- float형: 전체 피처 중 대상 피처의 퍼센트 (ex. max_features = 0.1)
- 'sqrt' = 'auto': (전체 피처 개수)$^{1/2}$(ex. max_features = 'auto')
- 'log2' :$\log_2$(전체 피처 개수) (ex. max_features = 'log2')

총 피처 수 49개
max_features = 'sqrt'로 지정
-> split 할 때 마다 random 하게 7개의 피처 선택

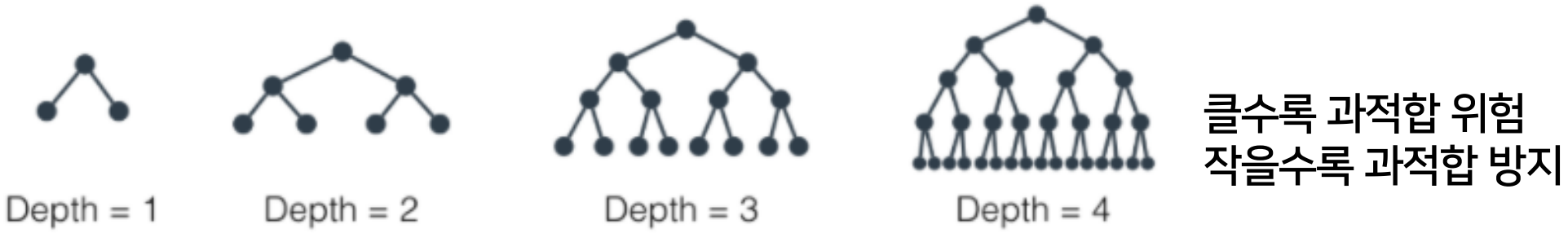
클수록: 모든 특성을 고려. 각 tree들이 매우 비슷
→가장 두드러진 특성을 이용해 데이터 맞춤
작을수록 : 각 tree들이 많이 달라짐 tree의 깊이가 깊어짐

결정 트리 파라미터

max_depth

- 트리의 최대 깊이 규정
- Default = None
  - Min_samples_split 설정대로 최대 분할
  - 과적합 위험

⇒ 적절한 값으로 제어 필요



Maximum depth of a decision tree

클수록 과적합 위험
작을수록 과적합 방지

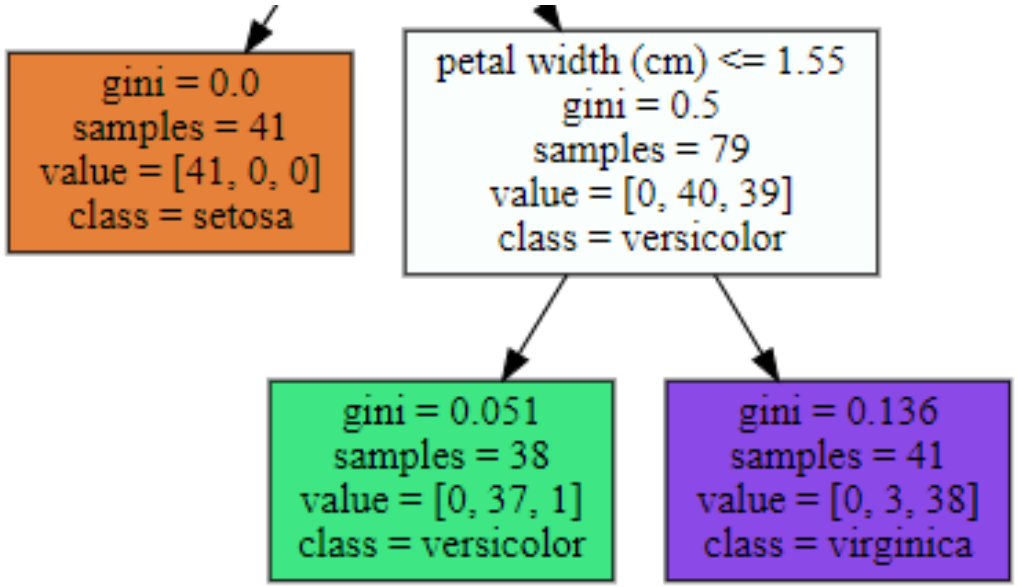
결정 트리 파라미터

max_leaf_nodes

말단 노드의 최대 개수

ex) max_leaf_nodes = 3로 설정
-> 말단 노드의 개수가 3개

클수록 과적합 위험
작을수록 과적합 방지



4. 시각화

Graphviz

- Graphviz

- 결정 트리 알고리즘이 어떤 규칙으로 트리를 생성하는지 시각적으로 보여주는 패키지
- dot파일로 기술된 다양한 이미지를 쉽게 시각화.
- 사이킷런은 graphviz와 인터페이스하기 위해 export_graphviz() API를 제공.

* export_graphviz(학습 완료된 estimator, feature_names, label_names)
→학습된 결정 트리를 실제 트리로 시각화

Graphviz

- 설치 방법(하양이 책 190쪽 참조!)

1. https://graphviz.org/download/ 에서 graphviz-2.38 msi 다운로드 후 설치
2. Anaconda Prompt에서 pip install graphviz 실행
3. 내 PC->속성->고급 시스템 설정->환경변수에서
   사용자 변수 Path에 C:₩Program files(x86)₩graphviz₩bin
   시스템 변수 Path에 C:₩Program files(x86)₩graphviz₩bin₩dot.exe 지정

3.의 과정에서 오류가 발생한다면 같은 코드를 google colab으로 수행하면 별도의 변수 지정 없이도 graphviz 사용 가능

Graphviz

- Graphgviz 쉽게 다운 받을 수 있는 페이지
- https://www.npackd.org/p/org.graphviz.Graphviz/2.38

- Graphviz 설치하는 방법 소개한 블로그 게시물
- https://blog.naver.com/sunmo9/222078545367

Graphviz를 통한 iris 데이터 결정 트리 시각화

```
#iris 데이터를 활용하여 decisiontreeclassifier로 학습.
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
dt_clf=DecisionTreeClassifier(random_state=24)
iris=load_iris()
iris_data=iris.data
iris_label=iris.target
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_label,test_size=0.2,random_state=23)
dt_clf.fit(x_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=24, splitter='best')
```

Iris 데이터를 feature와 label로 분리
->train, test데이터로 분리
->train data를 활용해 결정트리 모델로 학습

Graphviz를 통한 iris 데이터 결정 트리 시각화

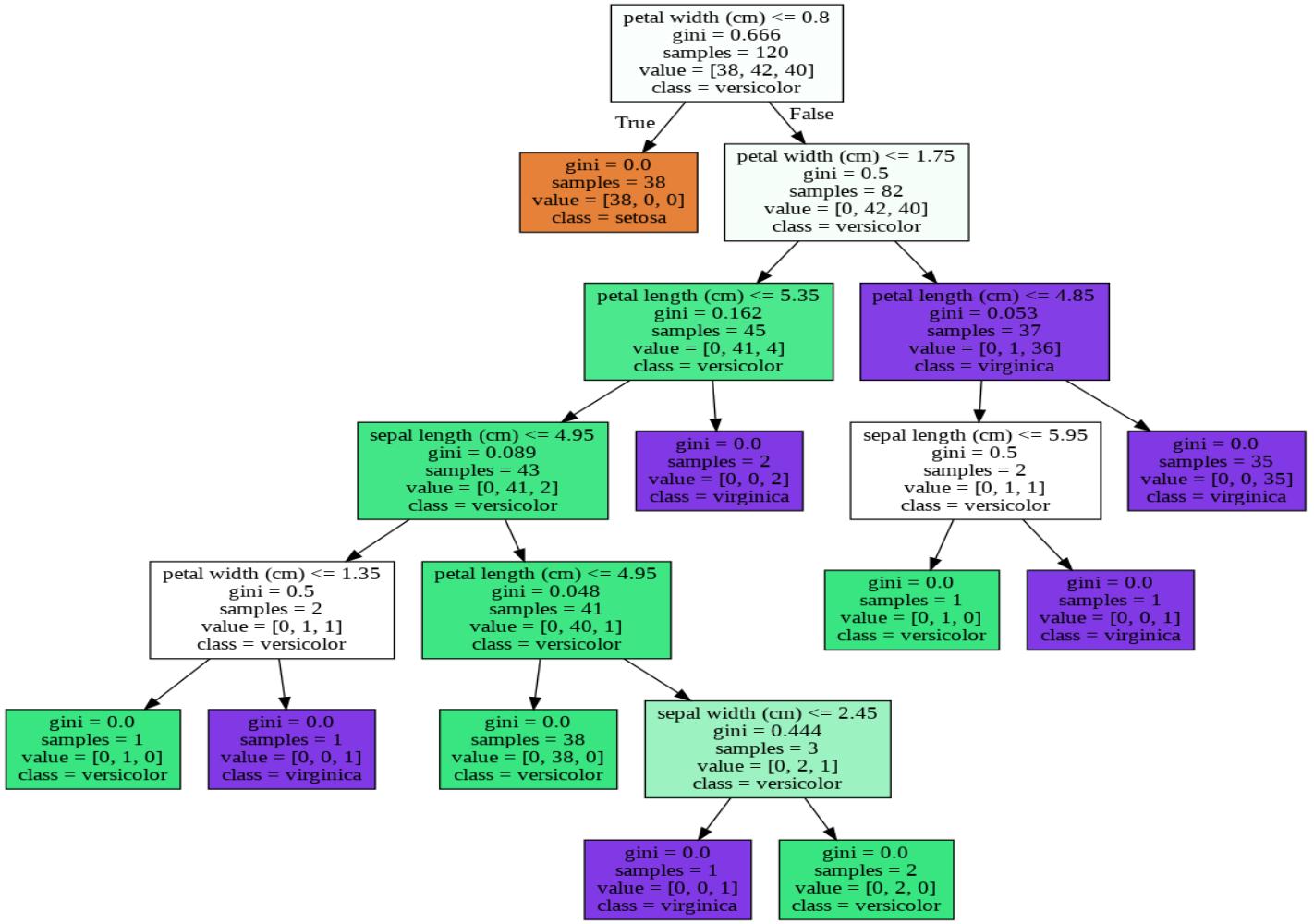
```
from sklearn.tree import export_graphviz
export_graphviz(dt_clf,out_file='tree.dot',class_names=iris.target_names,feature_names=iris.feature_names,impurity=True,filled=True)
import graphviz
with open('tree.dot') as f:
    dot_graph=f.read()
graphviz.Source(dot_graph)
```

export graphviz수행의 결과물(out_file)을 tree.dot으로 저장.
→ graphviz 모듈을 import
→ dot_graph에 앞서 저장한 tree.dot을 할당
→ graphviz모듈로 시각화

*impurity는 자료의 균일도 표시 관련 인자
*fill은 각 노드의 색깔 표시 관련 인자(후에 설명!)

Graphviz를 통한 iris 데이터 결정 트리 시각화

* 시각화 결과



각 규칙에 따라 결정 트리의 브랜치 노드와 리프 노드가 어떻게 구성되어 있는지 한 눈에 알 수 있다.

*리프 노드
:최종 클래스 값이 결정되는 노드
하나의 클래스로만 데이터가 구성 or 리프 노드가 될 수 있는 하이퍼 파라미터의 조건을 충족

Graphviz를 통한 iris 데이터 결정 트리 시각화

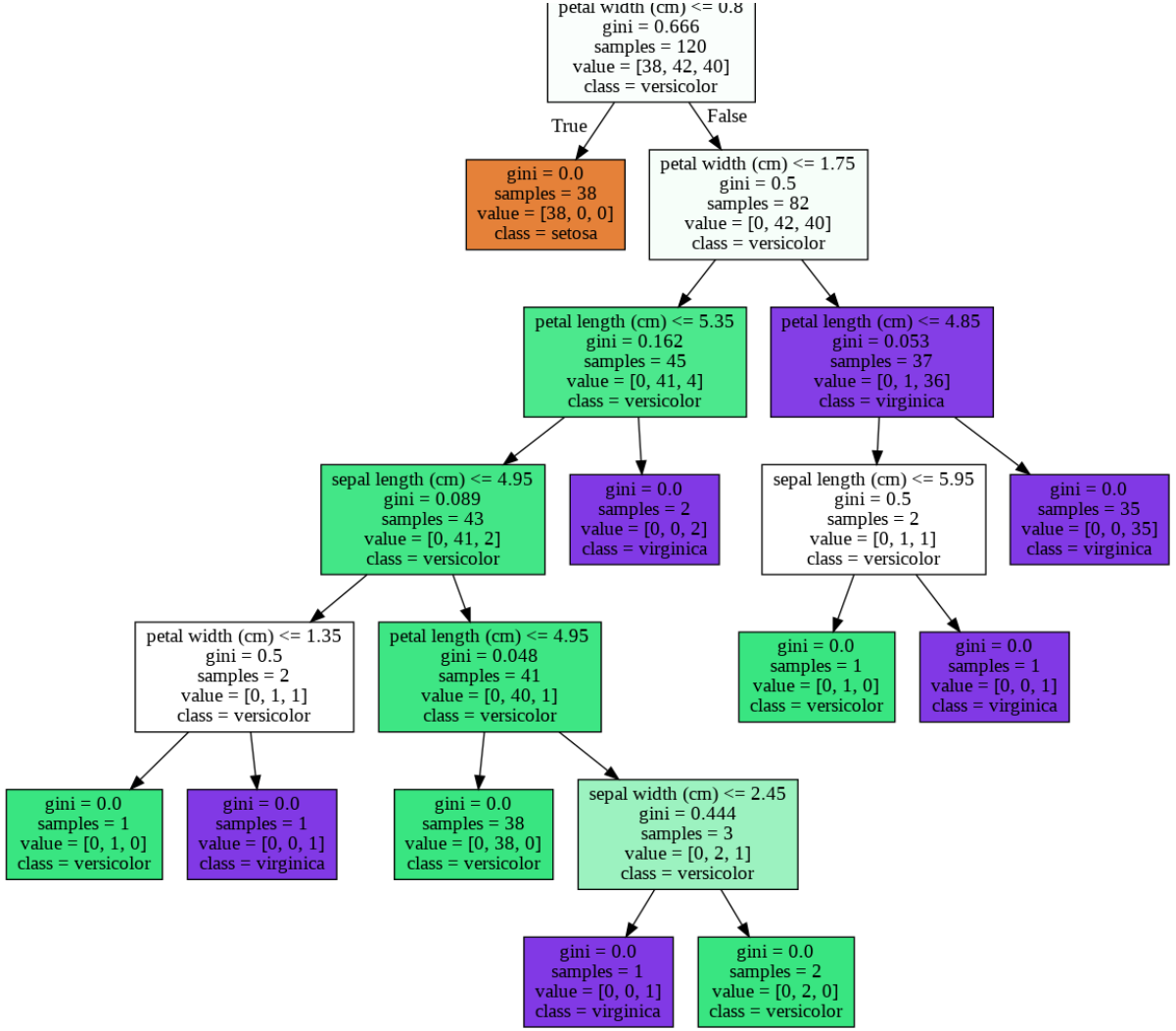
- 결정 트리 모형 분석

1. 규칙 노드 최상단에는 조건이 기술
2. Gini는 지니계수값
3. Samples에는 현 규칙에 해당하는 데이터 수
4. Value에는 클래스 값 기반의 데이터 수가 리스트 표시
5. 리프 노드의 class에는 분류 결과 class표시.
   규칙 노드의 class에는 많은 수의 데이터 class표시.
6. 각 노드의 색깔은 데이터의 레이블 값을 의미

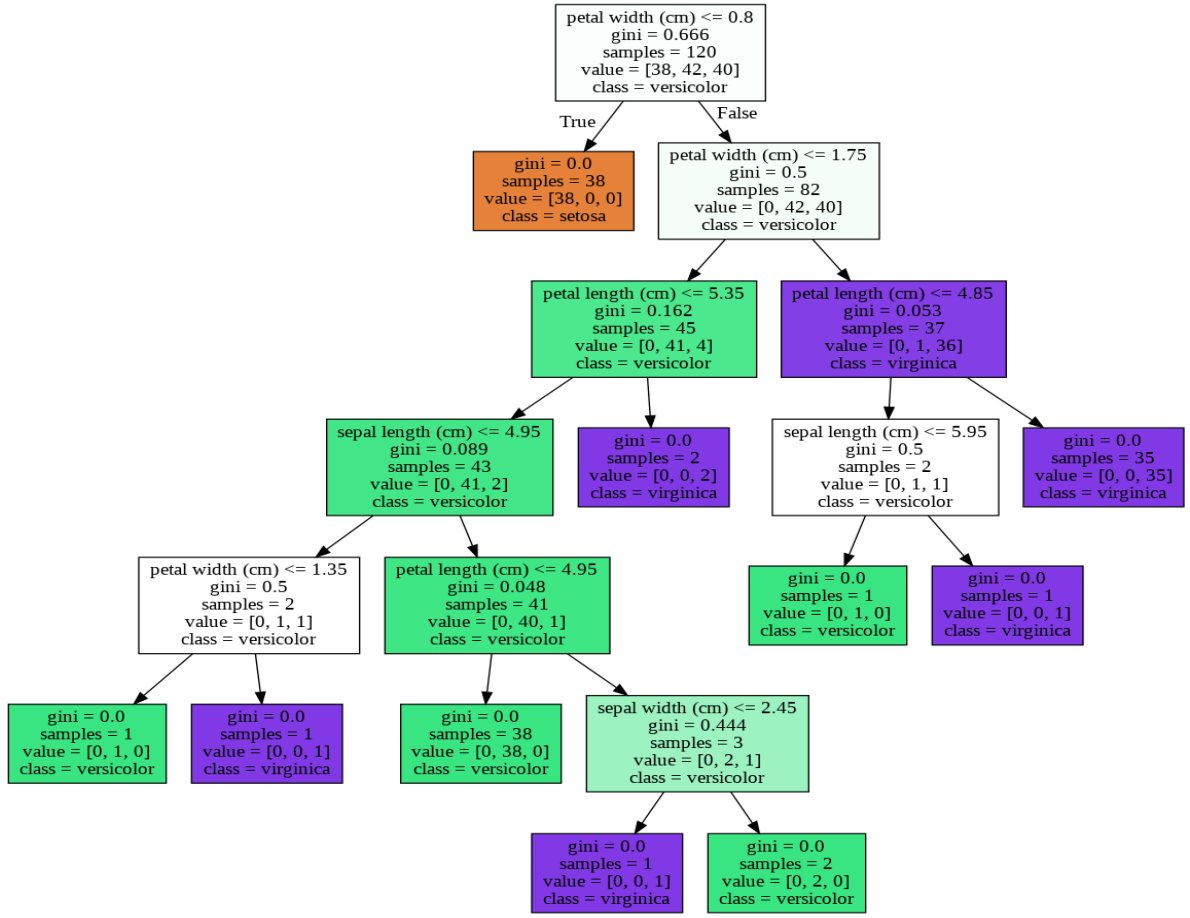
주황색-setosa, 초록색-versicolor, 보라색-virginica

→색깔이 짙어 질수록 지니계수가 낮고 해당 레이블에 속하는 샘플 데이터가 많다는 의미.

*노드 안에 기술된 지표를 통해 결정 트리를 쉽게 해석할 수 있다!



Graphviz를 통한 iris 데이터 결정 트리 시각화



현재 결정 트리의 하이퍼 파라미터로 아무 것도 지정하지 않은 상태

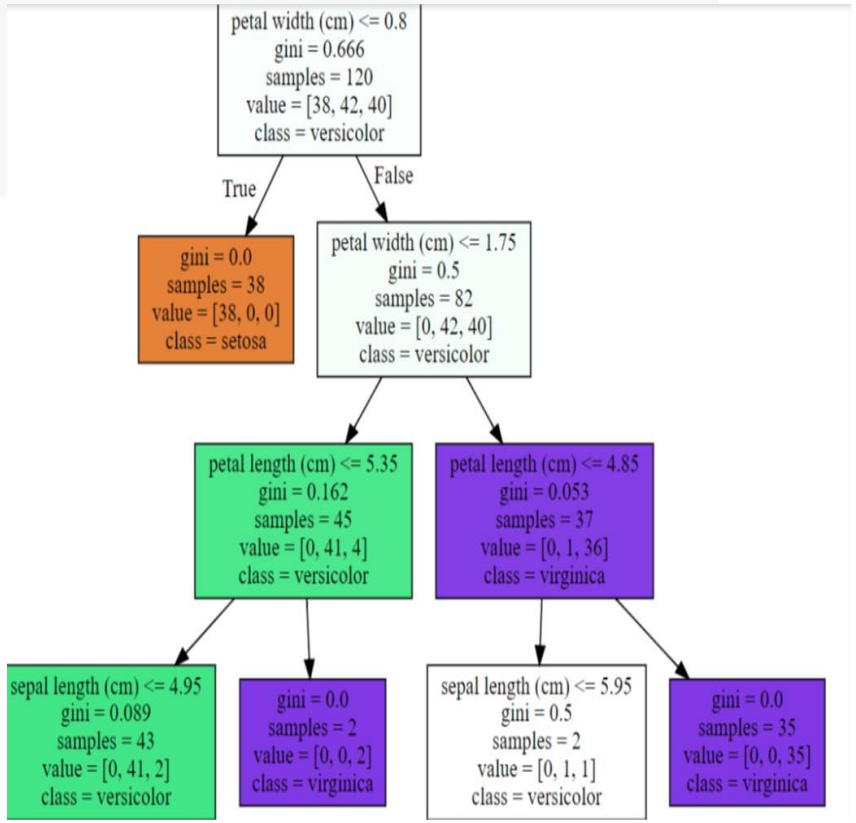
*첫번째 보라색 규칙 노드
-Vesicolor 데이터가 하나 뿐이지만 virginica와 구분하기 위해 불필요하게 자식 노드 생성

-하이퍼 파라미터를 지정하여 규칙 생성 로직을 제어하지 않으면 완벽하게 클래스 값을 구별하기 위해 자식 노드를 계속 생성→과적합의 문제 발생!

Graphviz를 통한 iris 데이터 결정 트리 시각화

* 과적합 문제 해결

```
dt_clf_max=DecisionTreeClassifier(max_depth=3).fit(x_train,y_train)
export_graphviz(dt_clf_max,out_file='tree.dot_max',class_names=iris.target_names,feature_names=iris.feature_names,impurity=True,filled=True)
with open('tree.dot_max') as d:
    dot_graph2=d.read()
graphviz.Source(dot_graph2)
```

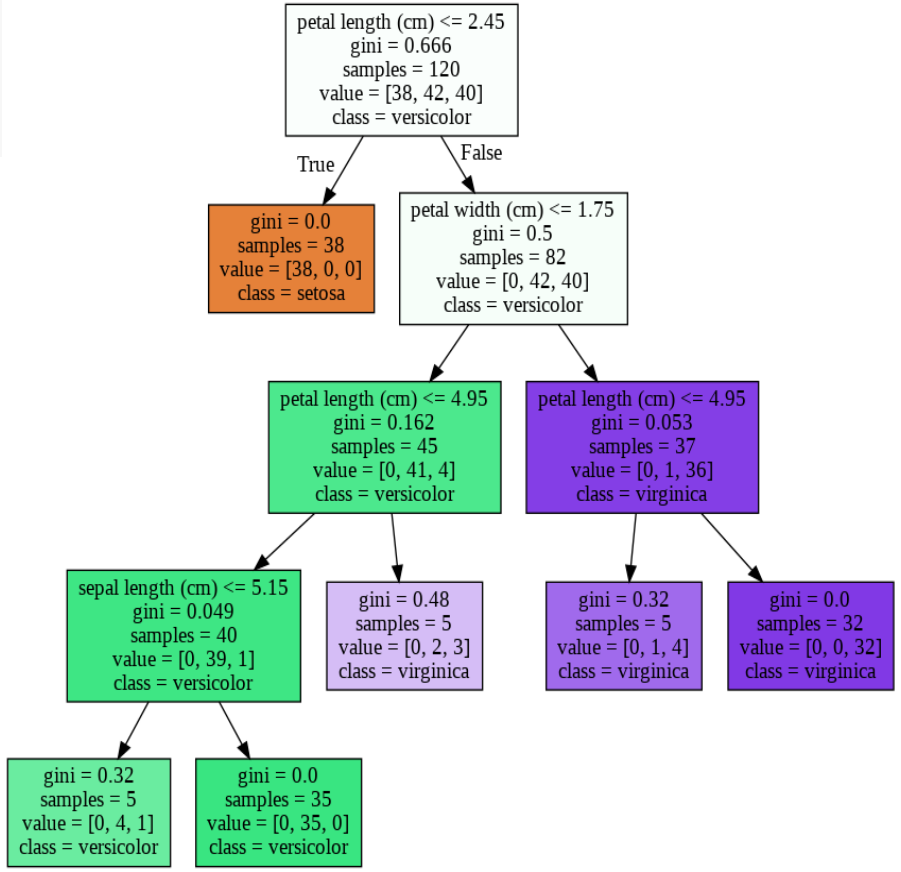


1.max_depth지정
결정 트리의 최대 깊이를 제어하면 더 간단한 결정 트리 만들어짐->과적합 문제 해결.

Graphviz를 통한 iris 데이터 결정 트리 시각화

* 과적합 문제 해결

```
dt_clf_min_leaf=DecisionTreeClassifier(min_samples_leaf=5).fit(x_train,y_train)
export_graphviz(dt_clf_min_leaf,out_file='tree.dot_min_leaf',class_names=iris.target_names,feature_names=iris.feature_names,impurity=True,filled=True)
with open('tree.dot_min_leaf') as d:
    dot_graph3=d.read()
graphviz.Source(dot_graph3)
```



2. min_samples_leaf 지정
 리프 노드가 될 수 있는 샘플 데이터 건수의 최소값 지정

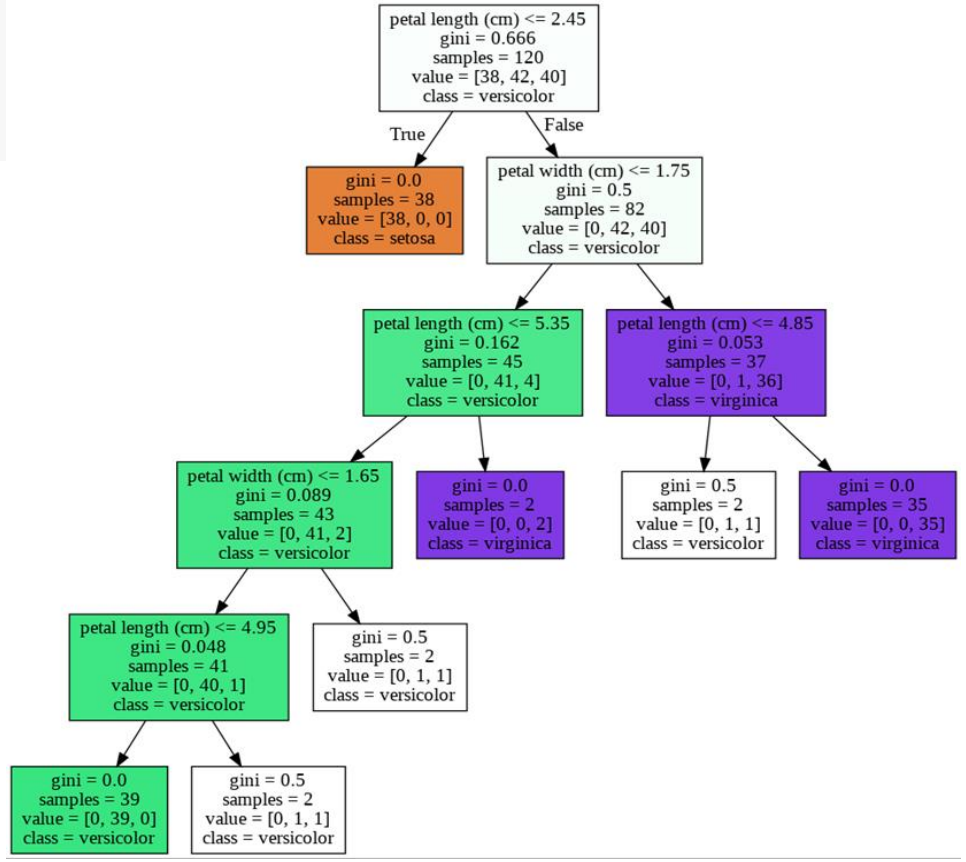
Default는 1→샘플 데이터 건수가 하나여야 리프 노드가 됨
→과적합 문제 발생!

*노드의 샘플 데이터 건수가 5 이하 이면 리프 노드가 된다
->과적합 문제 해결!

Graphviz를 통한 iris 데이터 결정 트리 시각화

* 과적합 문제 해결

```
dt_clf_min_split=DecisionTreeClassifier(min_samples_split=5).fit(x_train,y_train)
export_graphviz(dt_clf_min_split,out_file='tree.dot_min_split',class_names=iris.target_names,feature_names=iris.feature_names,impurity=True,filled=True)
with open('tree.dot_min_split') as d:
    dot_graph4=d.read()
graphviz.Source(dot_graph4)
```



3. min_samples_split 지정
 자식 노드를 생성하기 위한 샘플 데이터 건수의 최소값 지정

Default는 2 → 샘플 데이터가 2개여도 자식 노드 생성
→ 샘플 데이터 건수가 1이 될 때 까지 자식 노드 계속 생성
→ 과적합 문제 발생!

- 노드의 샘플 데이터 건수가 5보다 작으면 자식 노드 생성을 멈춤

-> 과적합 문제 해결!

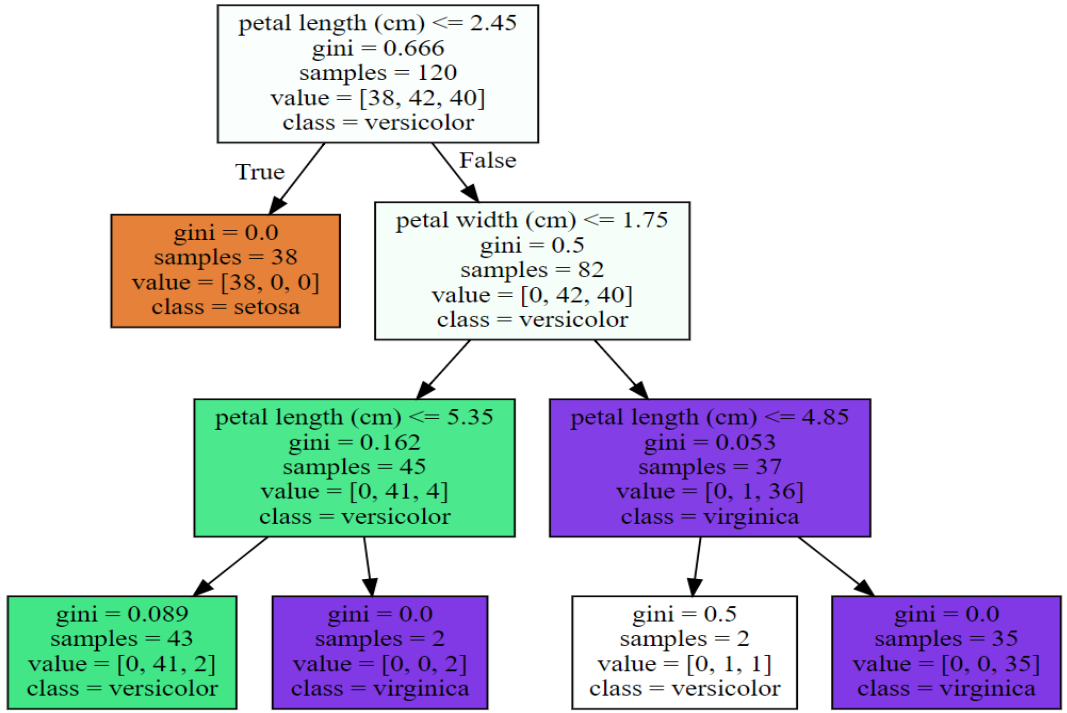
Graphviz를 통한 iris 데이터 결정 트리 시각화

* 과적합 문제 해결

```
dt_clf_max_leaf=DecisionTreeClassifier(max_leaf_nodes=5).fit(x_train,y_train)
export_graphviz(dt_clf_max_leaf,out_file='tree.dot_max_leaf',class_names=iris.target_names,feature_names=iris.feature_names,impurity=True,filled=True)
with open('tree.dot_max_leaf') as d:
    dot_graph5=d.read()
graphviz.Source(dot_graph5)
```

4. max_leaf_nodes 지정
 :최대 리프 노드 수 지정

→ 리프 노드 수가 해당 값에 도달하면 자식 노드 생성 멈춤 → 과적합 문제 해결



Graphviz를 통한 iris 데이터 결정 트리 시각화

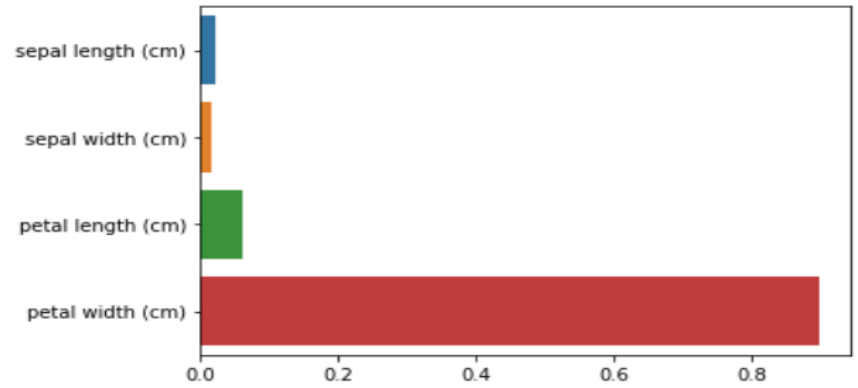
* 데이터 피처 중요도 파악

중요한 피처→명확한 결정 트리 만드는데 크게 기여

-DecisionTreeClassifierr객체의 feature_importances_속성으로 피처의 중요도를 파악할 수 있다.

```
import seaborn as sns
import numpy as np
%matplotlib inline
print('Feature importances: {0}'.format(np.round(dt_clf.feature_importances_,3)))
for name, value in zip(iris.feature_names,dt_clf.feature_importances_):
  print('{0} : {1:.3f}'.format(name,value))
sns.barplot(x=dt_clf.feature_importances_,y=iris.feature_names)
```

```
Feature importances: [0.023 0.017 0.063 0.897]
sepal length (cm) : 0.023
sepal width (cm) : 0.017
petal length (cm) : 0.063
petal width (cm) : 0.897
<matplotlib.axes._subplots.AxesSubplot at 0x7f51ee064208>
```



학습된 모델의 Feature_importances_속성은 ndarray형태로 값을 반환하고 피처 순서대로 값이 할당된다.

결정 트리 과적합 관련 시각화

• 분류를 위한 테스트 데이터 만들기

Make_classification함수를 이용해서 3개 클래스, 2개 피쳐를 가진 실험 데이터 만들기! (클래스 별로 색깔구분)

```
#분류를 위한 테스트 데이터 만들기
from sklearn.datasets import make_classification
x_features, y_labels=make_classification(n_features=2,n_redundant=0,n_informative=2,n_classes=3,n_clusters_per_class=1,random_state=0)
```

- n_samples : 표본 데이터의 수 (default=100)
- n_features : 독립 변수의 수(전체 피처의 수) (default=20)
- n_informative : 독립 변수 중 종속 변수와 상관 관계가 있는 성분의 수 (default=2)
- n_redundant : 독립 변수 중 다른 독립 변수의 선형 조합으로 나타나는 성분의 수 (default=2)
- n_repeated : 독립 변수 중 단순 중복된 성분의 수 (default=0)
- n_classes : 종속 변수의 클래스 수 default=2)
- n_clusters_per_class : 클래스 당 클러스터의 수 (default=2)
- weights : 각 클래스에 할당된 표본 수 (default=None)
- flip_y : 클래스가 임의로 교환되는 샘플의 일부, 라벨에 노이즈를 생성하여 분류를 어렵게 만든다(default=0.01)

x_features->표준정규분포에서 생성된 난수가 피처 당 100개 생성.
y_labels->각 피처 내 데이터의 클래스로 지정될 난수가 100개 생성.

```
print('클래스 값:', np.unique(y_labels))
print(len(y_labels))
print(type(y_labels))
```

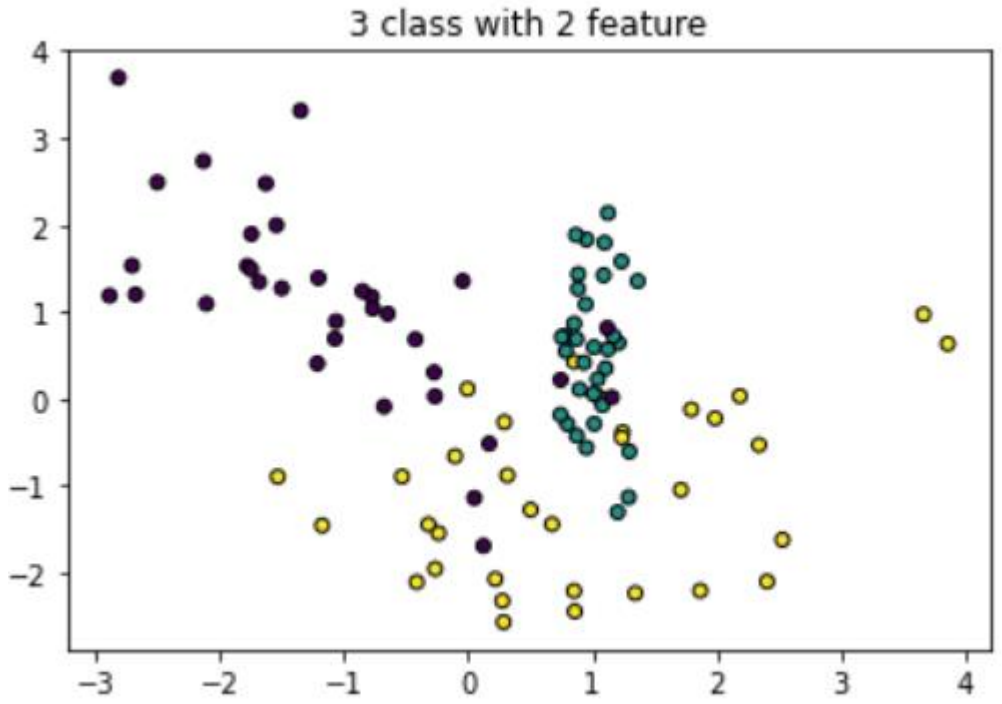
```
클래스 값: [0 1 2]
100
<class 'numpy.ndarray'>
```

```
print(x_features[0:5,:])
print(type(x_features))
print(x_features.shape)
```

```
[[ 0.12041909 -1.68941375]
 [ 1.09665605  1.79445113]
 [ 0.88426577  1.43370121]
 [ 0.95048011 -0.56202253]
 [ 1.04421447  0.02899023]]
<class 'numpy.ndarray'>
(100, 2)
```

결정 트리 과적합 관련 시각화

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.title('3 class with 2 feature')
plt.scatter(x_features[:,0],x_features[:,1],marker='o',c=y_labels,s=25,edgecolor='k')
```

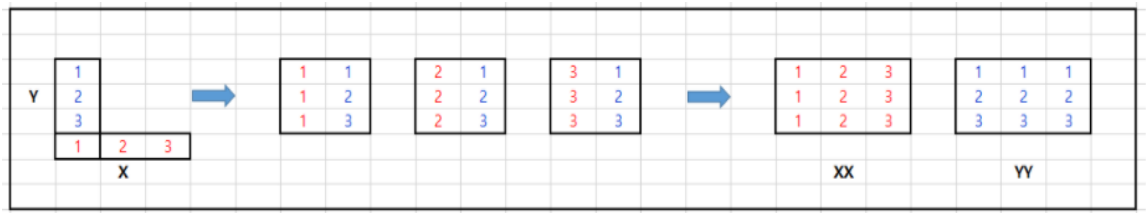


클래스 별로 색깔 구분해서 산점도 그려보기
→ 의도하는 바 대로 테스트 데이터가 만들어졌음을 알 수 있다.

결정 트리 과적합 관련 시각화

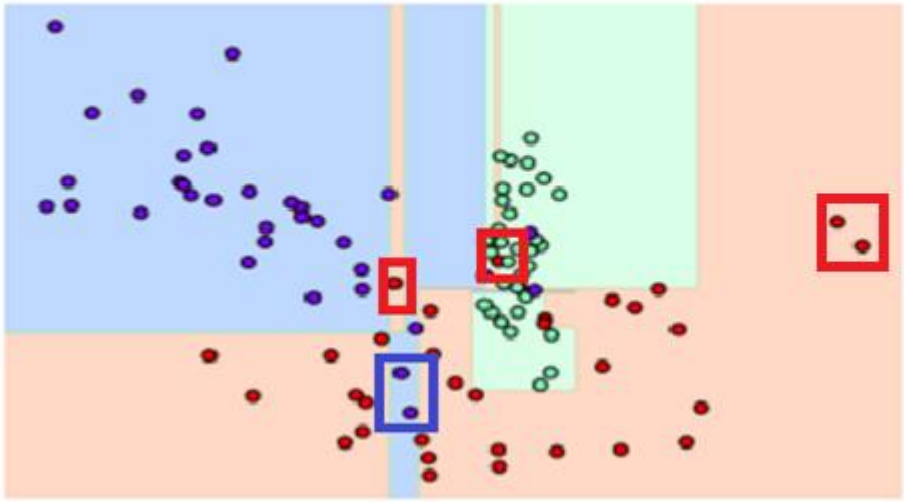
*시각화 함수 정의(colormap시각화)->하양이 예제 코드 참조

```
"# Classifier의 Decision Boundary를 시각화 하는 함수₩n",
def visualize_boundary(model, x, y):
  fig,ax = plt.subplots()
  # 학습 데이터 scatter plot으로 나타내기
  ax.scatter(x[:, 0], x[:, 1], c=y, s=25, cmap='rainbow', edgecolor='k',
             clim=(y.min(), y.max()), zorder=3) #z-order는 만들어지는 창의 개수
  ax.axis('tight') #모든 데이터를 볼 수 있도록 축의 범위를 설정.
  ax.axis('off')   #축과 라벨 끄기
  xlim_start , xlim_end = ax.get_xlim()
  ylim_start , ylim_end = ax.get_ylim()
  # 호출 파라미터로 들어온 training 데이터로 model 학습
  model.fit(x, y)
  # meshgrid 형태인 모든 좌표값으로 예측 수행.
  xx, yy = np.meshgrid(np.linspace(xlim_start,xlim_end, num=200),np.linspace(ylim_start,ylim_end, num=200)) #그림 설명
  Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape) #모델로 예측 수행 후 xx의 shape로 reshape.
  # contourf() 를 이용하여 class boundary 를 visualization 수행
  n_classes = len(np.unique(y)) #클래스 개수
  contours = ax.contourf(xx, yy, Z, alpha=0.3,
                         levels=np.arange(n_classes + 1) - 0.5,#클래스에 따라 등고선 개수, 위치 지정
                         cmap='rainbow', clim=(y.min(), y.max()))
```



하이퍼 파라미터가 default일시 클래스 결정 기준 경계

```
dt_clf=DecisionTreeClassifier().fit(x_features,y_labels)
visualize_boundary(dt_clf,x_features,y_labels)
```

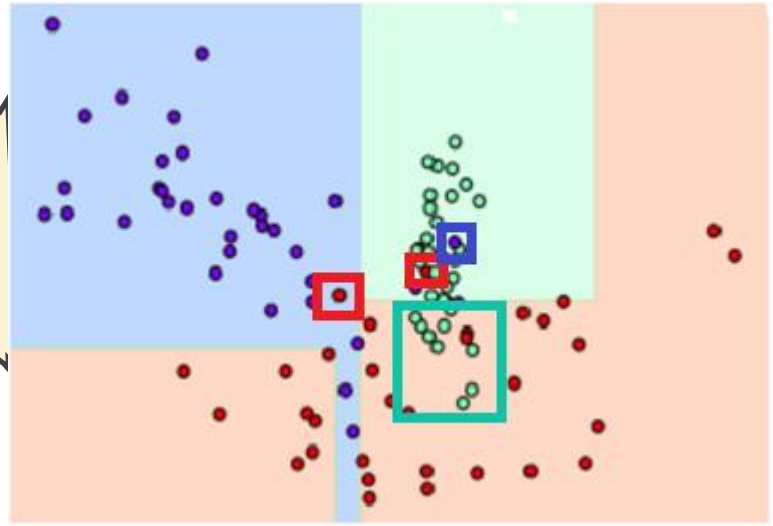


일부 이상치 데이터까지 분류하기 위해 과적합이 발생
→ 결정 기준 경계가 매우 많아지고 복잡해짐
→ 과적합 발생, 예측 정확도 떨어짐.

결정 트리 과적합 관련 시각화

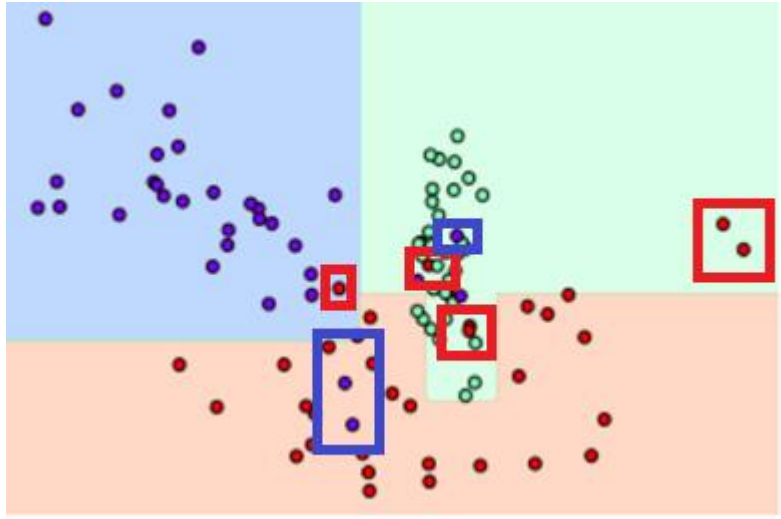
Max_depth=3일시 결정 기준 경계

```
dt_clf_max=DecisionTreeClassifier(max_depth=3).fit(x_features,y_labels)
visualize_boundary(dt_clf_max,x_features,y_labels)
```



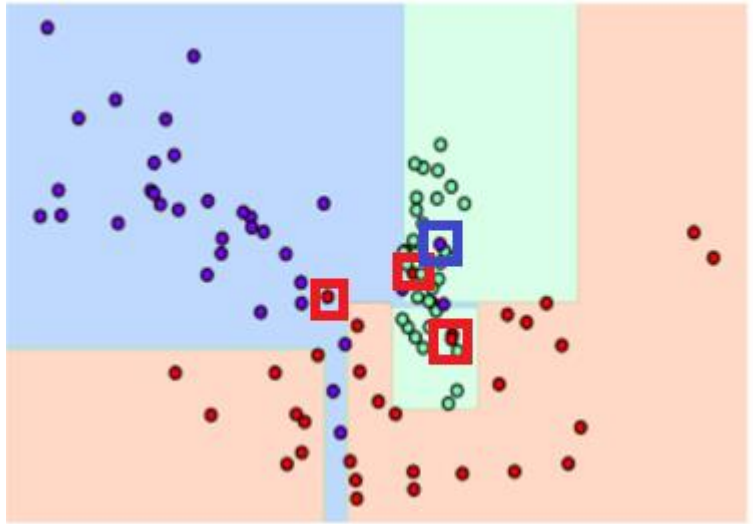
Min_samples_leaf=5일시 결정 기준 경계

```
dt_clf_min_leaf=DecisionTreeClassifier(min_samples_leaf=5).fit(x_features,y_labels)
visualize_boundary(dt_clf_min_leaf,x_features,y_labels)
```



Min_samples_split=5일시 결정 기준 경계

```
dt_clf_min_split=DecisionTreeClassifier(min_samples_split=5).fit(x_features,y_labels)
visualize_boundary(dt_clf_min_split,x_features,y_labels)
```



하이퍼 파라미터 제어→간단한 결정 트리→일부 이상치 데이터 분류하기 위해 규칙 노드를 더 생성하지 않음→과적합 문제 해결

하이퍼 파라미터를 조정한 모델이 더욱 뛰어난 분류 작업을 수행하고 예측 성능도 높음을 알 수 있다.

Iris 데이터 결정 트리의 최적 파라미터 찾아보기

1. GridSearchCV를 활용해서 결정 트리의 최적 파라미터 탐색.

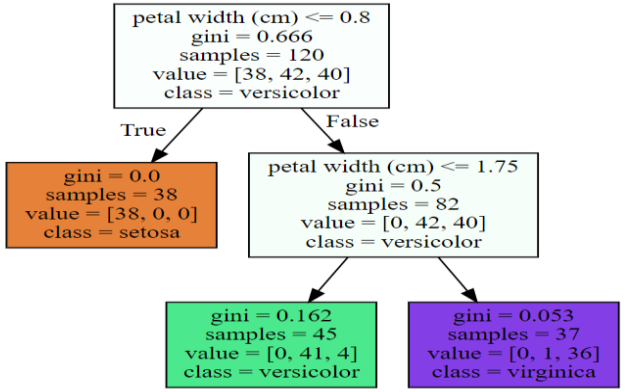
```
from sklearn.model_selection import GridSearchCV
parameters={'max_depth':[1,2,3,4,5,6],'min_samples_split':[1,2,3,4,5,6],'min_samples_leaf':[1,2,3,4,5,6],'max_leaf_nodes':[1,2,3,4,5,6]}
dtree=DecisionTreeClassifier()
grid_dtree=GridSearchCV(dtree,param_grid=parameters,cv=3,refit=True)
grid_dtree.fit(x_train,y_train)
print('최적 하이퍼 파라미터: {0}'.format(grid_dtree.best_params_))
```

```
최적 하이퍼 파라미터: {'max_depth': 2, 'max_leaf_nodes': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
{'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

2. 최적 파라미터로 모델 구축->시각화

```
dt_clf_best=DecisionTreeClassifier(max_depth=2, max_leaf_nodes=3, min_samples_leaf=1, min_samples_split=2).fit(x_train,y_train)
export_graphviz(dt_clf_best,out_file='best.dot',class_names=iris.target_names,feature_names=iris.feature_names,impurity=True,filled=True)
with open('best.dot') as d:
    dot_graph=d.read()
graphviz.Source(dot_graph)
```



5. 결정 트리 실습
- 사용자 행동 인식 데이터 세트

결정 트리 실습 – 사용자 행동 인식 데이터 세트

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# features.txt파일에는 피처 이름 index와 피처명이 공백으로 분리되어있음.
# 이를 DataFrame으로 로드
feature_name_df = pd.read_csv('./human_activity/features.txt',
                              sep='₩s+',header=None,
                              names=['column_index','column_name'])

# 피처명 index를 제거하고 피처명만 리스트 객체로 생성한 뒤 샘플로 10개만 추출
feature_name=feature_name_df.iloc[:,1].values.tolist()
print('전체 피처명에서 10개만 추출 : ',feature_name[:10])
```

```
전체 피처명에서 10개만 추출 :  ['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z', 'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z', 'tBodyAcc-max()-X']
```

종류	설명
₩w	유니코드인 경우 숫자, 밑줄(underscore, '_')를 포함하는 모든 언어의 표현 가능한 문자, 아스키코드인 경우 '[a-zA-Z0-9_]'와 동일함
₩W	유니코드인 경우 숫자, 밑줄과 표현 가능한 문자를 제외한 나머지 문자, 아스키코드인 경우 '[^a-zA-Z0-9_]'와 동일함
₩d	유니코드인 경우 [0-9]를 포함하는 모든 숫자임. 아스키코드인 경우 [0-9]와 동일함
₩D	유니코드인 경우 숫자를 제외한 모든 문자 아스키코드인경우 [^0-9]와 동일함
₩s	유니코드인경우 [₩t₩n₩r₩f₩v]를 포함하는 공백문자 아스키코드인경우 [₩t₩n₩r₩f₩v]와 동일함
₩S	유니코드인 경우 공백문자를 제외한 모든 문자 아스키코드인경우 [^₩t₩n₩r₩f₩v]와 동일함
₩b	단어의 시작과 끝의 빈공백
₩B	단어의 시작과 끝이 아닌 빈공백
₩₩	역슬래시(₩) 문자 자체를 의미
₩[숫자]	지정된 숫자만큼 일치하는 문자열을 의미
₩A	문자열의 시작
₩Z	문자열의 끝

```
feature_name_df
```

	column_index	column_name
0	1	tBodyAcc-mean()-X
1	2	tBodyAcc-mean()-Y
2	3	tBodyAcc-mean()-Z
3	4	tBodyAcc-std()-X
4	5	tBodyAcc-std()-Y
...	...	...
556	557	angle(tBodyGyroMean,gravityMean)
557	558	angle(tBodyGyroJerkMean,gravityMean)
558	559	angle(X,gravityMean)
559	560	angle(Y,gravityMean)
560	561	angle(Z,gravityMean)

561 rows × 2 columns

Sep='₩s+'
정규표현식 형태로 분리 단위 표현한 것
+ : 1회 이상 반복
=>공백 기준 분리

결정 트리 실습 – 사용자 행동 인식 데이터 세트

중복 피처명 처리
피처명 파일에는 중복 피처명이 존재하는데 이대로 DataFrame으로 로드하면 오류 발생 (과거 판다스 버전은 허용했지만 현재 버전이 허용하지 않음)

1. 중복된 피처명이 얼마나 있는지 파악하기

```
feature_dup_df = feature_name_df.groupby('column_name').count()
print(feature_dup_df[feature_dup_df['column_index']>1].count())
feature_dup_df[feature_dup_df['column_index']>1].head()
```

```
column_index    42
dtype: int64
```

총 42개의 피처명 중복

column_name	column_index
fBodyAcc-bandsEnergy()-1,16	3
fBodyAcc-bandsEnergy()-1,24	3
fBodyAcc-bandsEnergy()-1,8	3
fBodyAcc-bandsEnergy()-17,24	3
fBodyAcc-bandsEnergy()-17,32	3

결정 트리 실습 – 사용자 행동 인식 데이터 세트

2. 원본 피처명에 _1또는 _2를 추가로 부여해 새로운 피처명 가지는 DataFrame 반환하는 함수 생성

```
def get_new_feature_name_df(old_feature_name_df):
    feature_dup_df = pd.DataFrame(data=old_feature_name_df.groupby('column_name')
                                  .cumcount(),columns=['dup_cnt'])
    feature_dup_df = feature_dup_df.reset_index()
    new_feature_name_df = pd.merge(old_feature_name_df.reset_index(),
                                   feature_dup_df, how='outer')
    new_feature_name_df['column_name'] = new_feature_name_df[[
        'column_name', 'dup_cnt']].apply(lambda x : x[0]+'_'+str(x[1])
                                         if x[1] >0 else x[0] ,  axis=1)
    new_feature_name_df = new_feature_name_df.drop(['index'], axis=1)
    return new_feature_name_df
```

```
>>> df = pd.DataFrame([['a'], ['a'], ['a'], ['b'], ['b'], ['a']],
...                   columns=['A'])
>>> df
   A
0  a
1  a
2  a
3  b
4  b
5  a
>>> df.groupby('A').cumcount()
0    0
1    1
2    2
3    0
4    1
5    3
```

* 코드 안치셔도 됩니다

1. 기존 피처 명 DataFrame을 피처 명을 기준으로 그룹화 후 cumcount() 적용 값을 dup_cnt칼럼에 저장
2. 같은 피처 명의 경우 번호가 매겨진 DataFrame과 기존 DataFrame을 merge
3. 만약 dup_cnt 칼럼의 값이 0보다 큰 경우는 중복된 칼럼 명이기 때문에 dup_cnt의 값을 column_name 값에 '_1'의 형태로 추가

Groupby.cumcount()
각 그룹항목에 0부터 해당 길이까지의 번호 매김

결정 트리 실습 – 사용자 행동 인식 데이터 세트

```
# 데이터셋을 구성하는 함수 설정
def get_human_dataset( ):

    # 각 데이터 파일들은 공백으로 분리되어 있으므로 read_csv에서 공백 문자를 sep으로 할당.
    feature_name_df = pd.read_csv('./human_activity/features.txt',sep='₩s+',
                        header=None,names=['column_index','column_name'])

    # 중복된 피처명을 수정하는 get_new_feature_name_df()를 이용, 신규 피처명 DataFrame생성.
    new_feature_name_df = get_new_feature_name_df(feature_name_df)

    # DataFrame에 피처명을 컬럼으로 부여하기 위해 리스트 객체로 다시 변환
    feature_name = new_feature_name_df.iloc[:, 1].values.tolist()

    # 학습 피처 데이터 셋과 테스트 피처 데이터을 DataFrame으로 로딩. 컬럼명은 feature_name 적용
    X_train = pd.read_csv('./human_activity/train/X_train.txt',sep='₩s+', names=feature_name )
    X_test = pd.read_csv('./human_activity/test/X_test.txt',sep='₩s+', names=feature_name)

    # 학습 레이블과 테스트 레이블 데이터을 DataFrame으로 로딩하고 컬럼명은 action으로 부여
    y_train = pd.read_csv('./human_activity/train/y_train.txt',sep='₩s+',header=None,names=['action'])
    y_test = pd.read_csv('./human_activity/test/y_test.txt',sep='₩s+',header=None,names=['action'])

    # 로드된 학습/테스트용 DataFrame을 모두 반환
    return X_train, X_test, y_train, y_test


X_train, X_test, y_train, y_test = get_human_dataset()
```

Train 디렉터리의 학습용 피처 데이터 세트와 레이블 데이터 세트,
Test 디렉터이의 학습용 피처 데이터 세트와 레이블 데이터 세트
이용하여 DataFrame을 생성하는 함수 만들기

결정 트리 실습 – 사용자 행동 인식 데이터 세트

로드한 학습용 피처 데이터 세트 살펴보기

```
print('## 학습 피러 데이터셋 info()')
print(X_train.info())
```

```
## 학습 피러 데이터셋 info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 561 entries, tBodyAcc-mean()-X to angle(Z,gravityMean)
dtypes: float64(561)
memory usage: 31.5 MB
None
```

학습 데이터 세트
7352개의 레코드, 561개의 피처
피처는 전부 float형이라 별도의 카테고리 인코딩 수행 안해도 됨

```
print(y_train['action'].value_counts())
```

```
6    1407
5    1374
4    1286
1    1226
2    1073
3     986
Name: action, dtype: int64
```

레이블 값은 1~6의 6개
왜곡 없이 비교적 고르게 분포

결정 트리 실습 – 사용자 행동 인식 데이터 세트

사이킷런의 DecisionTreeClassifier 이용해 동작 예측 분류 문제 수행

```
from sklearn.metrics import accuracy_score

# 예제 반복시마다 동일한 예측 결과 도출을 위해 난수값(random_state) 설정
dt_clf = DecisionTreeClassifier(random_state=156)
dt_clf.fit(X_train, y_train)
pred = dt_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print('Decision Tree 예측 정확도 : {0:.4f}'.format(accuracy))

# DecisionTreeClassifier의 하이퍼 파리미터 추출
print('₩nDecisionTreeClassifier 기본 하이퍼파라미터:₩n', dt_clf.get_params())
```

```
Decision Tree 예측 정확도 : 0.8548

DecisionTreeClassifier 기본 하이퍼파라미터:
 {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1,
'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': 156,
'splitter': 'best'}
```

결정 트리 실습 – 사용자 행동 인식 데이터 세트

결정 트리의 Depth가 예측 정확도에 주는 영향 살펴보기

```
from sklearn.model_selection import GridSearchCV
params = {'max_depth' : [ 6, 8 ,10, 12, 16 ,20, 24]}
grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1)
grid_cv.fit(X_train , y_train)
print('GridSearchCV 최고 평균 정확도 수치:{0:.4f}'.format(grid_cv.best_score_))
print('GridSearchCV 최적 하이퍼 파라미터:', grid_cv.best_params_)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:  1.4min finished
```

```
GridSearchCV 최고 평균 정확도 수치:0.8526
GridSearchCV 최적 하이퍼 파라미터: {'max_depth': 8}
```

Max_depth가 16일 때 5개의 폴드 세트의 최고 평균 정확도 결과가 85.13%로 도출

```
#GridSearchCV객체의 cv_results_속성을 데이터 프레임으로 생성
cv_results_df = pd.DataFrame(grid_cv.cv_results_)
#max_depth 파라미터 값과 그때의 테스트 세트, 학습 데이터 세트의 정확도 수치 추출
cv_results_df[['param_max_depth', 'mean_test_score']]
```

	param_max_depth	mean_test_score
0	6	0.850925
1	8	0.852557
2	10	0.850925
3	12	0.844124
4	16	0.852149
5	20	0.851605
6	24	0.850245

Max_depth값의 증가에 따라 예측 성능이 어떻게 변했는지 확인
Cv_results_ 속성 : CV세트에 하이퍼 파라미터를 순차적으로 입력했을 때의 성능 수치
Mean_test_score는 5개의 CV 세트에서 검증용 데이터 세트 정확도 평균 수치

- GridSearchCV 내용 참조:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

결정 트리 실습 – 사용자 행동 인식 데이터 세트

테스트 데이터 세트에서 결정 트리의 정확도 측정

```
# GridSearch가 아닌 별도의 테스트 데이터셋에서 max_depth별 성능 측정
max_depths = [6, 8, 10, 12, 16, 20, 24]

for depth in max_depths:
    dt_clf = DecisionTreeClassifier(max_depth=depth, random_state=156)
    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_test)
    accuracy = accuracy_score(y_test, pred)
    print('max_depth = {0} 정확도 : {1:.4f}'.format(depth, accuracy))
```

```
max_depth = 6 정확도 : 0.8558
max_depth = 8 정확도 : 0.8707
max_depth = 10 정확도 : 0.8673
max_depth = 12 정확도 : 0.8646
max_depth = 16 정확도 : 0.8575
max_depth = 20 정확도 : 0.8548
max_depth = 24 정확도 : 0.8548
```

Max_depth가 8일 때 약 87.07%로 가장 높은 정확도
Max_depth가 8을 넘어가면서 정확도 감소

⇒ 깊이 깊어질수록 과적합의 영향력이 커짐
⇒ 하이퍼 파라미터를 이용해 깊이 제어 가능해야 함.
⇒ 복잡한 모델보다 트리 깊이를 낮춘 단순한 모델이 더 효과적인 결과일 수 있다.

결정 트리 실습 – 사용자 행동 인식 데이터 세트

Max_depth와 min_samples_split을 같이 변경하며 정확도 성능 튜닝

```
params = {
    'max_depth' : [6, 8, 10, 12, 16, 20, 24],
    'min_samples_split' : [16, 24]
}

grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5, verbose=1)
grid_cv.fit(X_train, y_train)
print('GridSearchCV 최고 평균 정확도 수치: {:.4f}'.format(grid_cv.best_score_))
print('GridSearchCV 최적 하이퍼파라미터: ', grid_cv.best_params_)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:  4.0min finished
```

```
GridSearchCV 최고 평균 정확도 수치: 0.8549
GridSearchCV 최적 하이퍼파라미터:  {'max_depth': 8, 'min_samples_split': 16}
```

Max_depth가 8, min_samles_split이 16일 때 최고 정확도 85.5%

```
best_df_clf = grid_cv.best_estimator_
pred1 = best_df_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred1)
print('Desicion Tree 예측 정확도: {0:.4f}'.format(accuracy))
```

```
Desicion Tree 예측 정확도: 0.8717
```

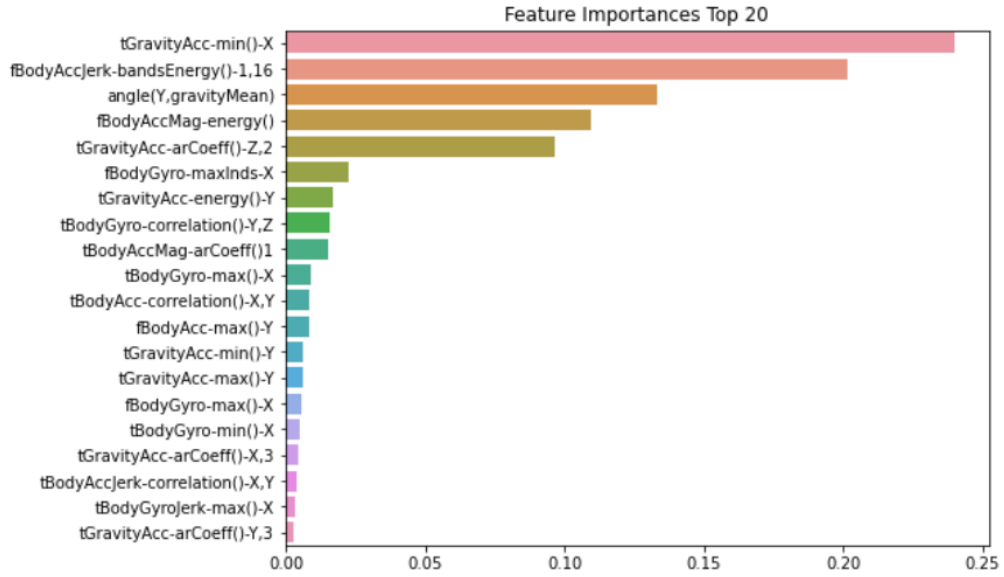
테스트 데이터 세트에 적용하면
Max_depth가 8, min_samles_split이 16일 때
최고 정확도 87.17%

결정 트리 실습 – 사용자 행동 인식 데이터 세트

결정 트리에서 각 피처의 중요도

```
import seaborn as sns

ftr_importance_values = best_df_clf.feature_importances_
# Top 중요도로 정렬하고, 쉽게 시각화하기 위해 Series 변환
ftr_importances = pd.Series(ftr_importance_values, index=X_train.columns)
# 중요도값 순으로 Series를 정렬
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
plt.figure(figsize=[8, 6])
plt.title('Feature Importances Top 20')
sns.barplot(x=ftr_top20, y=ftr_top20.index)
plt.show()
```



가장 높은 중요도 가진 top 5피처들이 중요하게 규칙 생성에 영향을 미침

Q&A

1조 – 김범준, 박소영, 오세정, 이현진