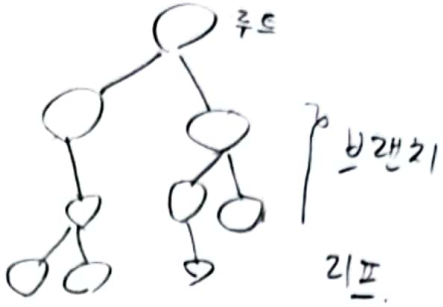
# 머신러닝 - 결정트리(분류)

## 결정트리

결정 트리는 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만드는 것이다.

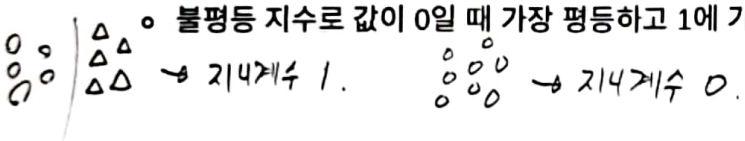



결정 트리는 다음와 같은 노드로 구성된다.

- 루트 노트(Root Node)
  - 초기 지점으로 첫 번째 분류 규칙 노드
- 브랜치 노드(Branch Node)
  - 자식 노드를 가지는 중간 규칙 노드로서 브랜치 노드, 리프 노드로 분리 될 수 있다.
- 리프 노드(Leaf Node)
  - 더 이상 자식 노드가 없는 노드로서 최종 클래스 값이 결정되는 노드

여기서 규칙은 정보 균일도가 높은 데이터 셋을 먼저 선택할 수 있도록 규칙조건을 만들며, 정보 균일도를 측정하는 대표 적인 방법은 지니 계수이다.

- 지니 계수
  - 불평등 지수로 값이 0일 때 가장 평등하고 1에 가까워질수록 불평등함




## 설치해야 할 라이브러리

- scikit-learn
  - pip install sklearn
- graphviz
  - conda install graphviz
  - 설치가 안되거나 라이브러리가 작동이 안 될경우 아래 링크 참조
  - https://moondol-ai.tistory.com/149

pip install graphviz

## 사용할 데이터셋(dataset)

데이터셋은 사이킷런 라이브러리에서 지원해주는 iris 데이터를 사용합니다.

붓꽃에 대한 데이터이며, 꽃잎의 각 부분의 너비와 길이를 측정한 데이터이다.

총 데이터의 수는 150개이며, 데이터의 형식은 아래와 같다.

| 이름 | 설명 |
|---|---|
| Sepal Length | 꽃받침의 길이 정보이다. |
| Sepal Width | 꽃받침의 너비 정도이다. |
| Petal Length | 꽃잎의 길이 정보이다. |
| Petal Width | 꽃잎의 너비 정보이다. |
| Species | 꽃의 종류 정보이다. (Setosa, Versicolor, Virginica) |

우리는 꽃받침과 꽃잎의 길이, 너비 정보를 사용하여 꽃이 어떤 종류에 속하는지 분류하는 모델을 구현한다.

from sklearn.datasets import load_iris

```
import warnings

warnings.filterwarnings("ignore") # 오류 무시 코드
```

머신러닝이나 딥러닝 코드 작업할 때는 사소한 오류들이 콘솔창에 많이 뜹니다. 오류들이 출력되지 않도록 코드를 입력합니다.

```
from sklearn.tree import DecisionTreeClassifier  # 결정트리 라이브러리.
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 사이킷런 라이브러리에 포함된 IRIS 데이터 로드
iris_data = load_iris()
# iris 데이터셋을 학습 데이터랑 테스트 데이터를 8:2 비율로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, 특성(feature)
                                                    iris_data.target, 정답(Label)
                                                    test_size=0.2,  train 80%. test 20%.
                                                    random_state=11) # 랜덤 시드

# 모델 적용
dt_clf = DecisionTreeClassifier(random_state=156)
```

```python
# 데이터 구성 확인
print("학습 데이터 개수 : " + str(len(X_train)))
print("테스트 데이터 개수 : " + str(len(X_test)))

# 모델 학습
dt_clf.fit(X_train, y_train) 훈련

# 학습 데이터 검증
pred = dt_clf.predict(X_train) 값 예측.
accuarcy = accuracy_score(y_train, pred) 정확도 계산
print("학습 데이터 검증 정확도 : ", accuarcy)

# 테스트 데이터 검증
pred = dt_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print("테스트 데이터 검증 정확도 : ", accuracy)
```

< 코드 2 출력결과 >

학습 데이터 개수 : 120

테스트 데이터 개수 : 30

학습 데이터 검증 정확도 : 1.0

테스트 데이터 검증 정확도 : 0.9333333333333333

```python
# 피처 별 중요도
feature_imp = dt_clf.feature_importances_ 특성의 중요도.

for name, value in zip(iris_data.feature_names, feature_imp):
    print(f"{name}: {value:.4f}")
```

< 코드 3 출력결과 >

sepal length (cm): 0.0250

sepal width (cm): 0.0000

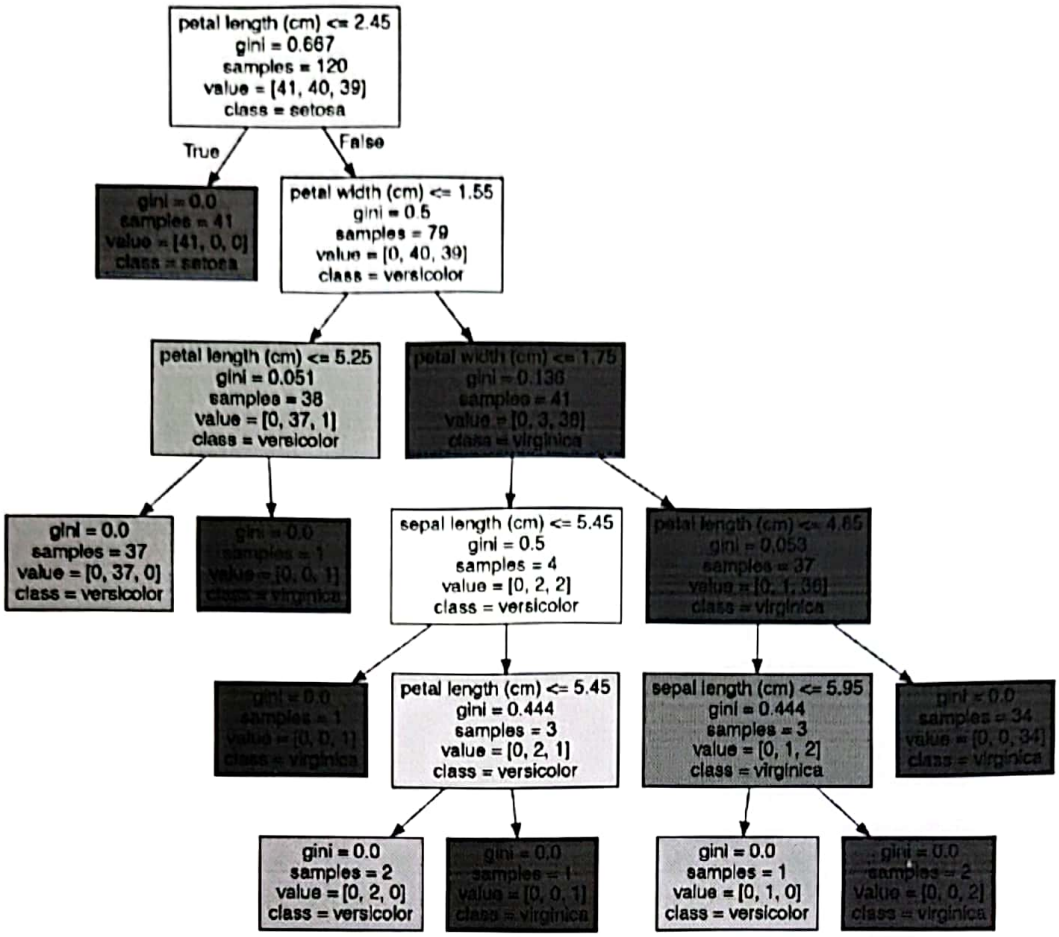
petal length (cm): 0.5549

petal width (cm): 0.4201

모델을 만드는데에는 성공하였지만, 정확히 어떠한 구조로 인하여 분류가 되는지는 정확히 알 수 없습니다. 트리형태를 시각화주는 라이브러리를 사용하여 모델의 구조를 확인해보겠습니다.

위에서 라이브러리를 설치하지 않았다면 돌아가서 설치 후 코드를 구현바랍니다.

```
from sklearn.tree import export_graphviz
import graphviz
# conda install graphviz
# graphviz 라이브러리에서 ExecutableNotFound 에러가 뜰 경우에는
# https://moondol-ai.tistory.com/149 참조해서 해결

export_graphviz(dt_clf, out_file="tree.dot",
                class_names= iris_data.target_names, # class 변수의 값
                feature_names = iris_data.feature_names, #
                impurity = True, # 지니 계수(불평등 지수) 출력
                filled= True) # 그림 색상 여부

with open("tree.dot") as f: # 저장한 파일 읽어오기
    dot_graph = f.read()

graphviz.Source(dot_graph)
```




코드 4 출력 결과

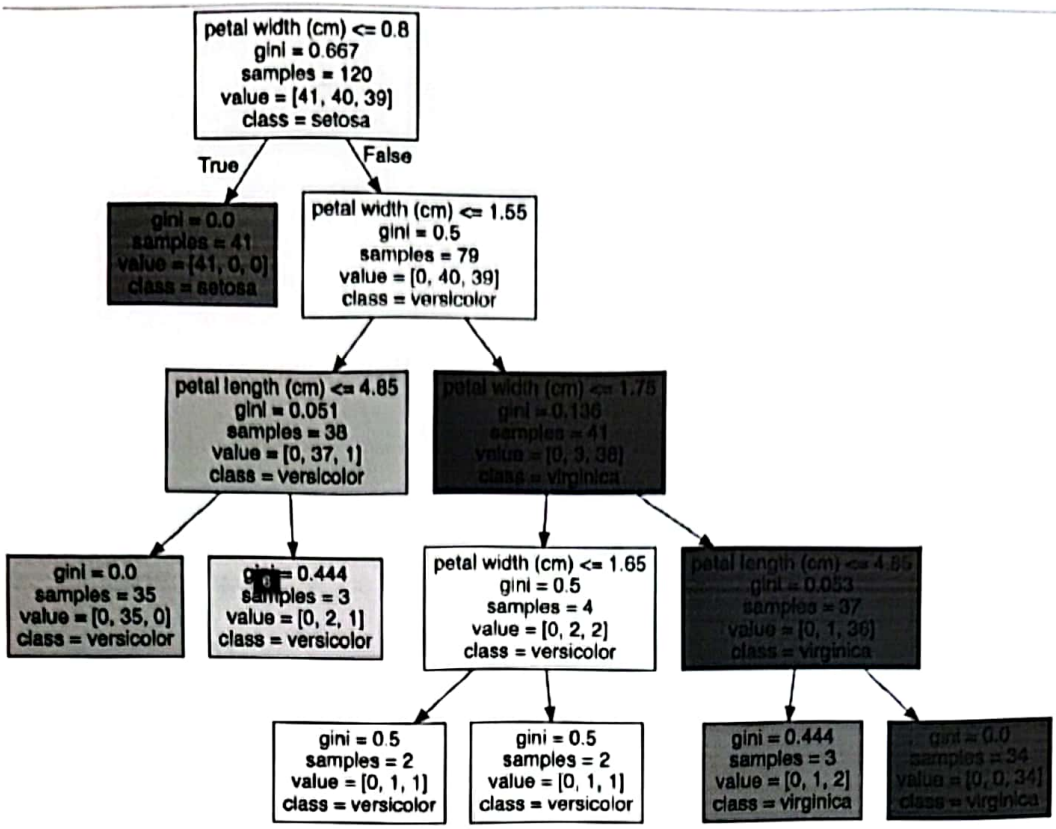
위의 리프노트부터 시작해서 하나의 value 안에 하나의 값만 나올때까지 계속해서 자식노드를 생성하는 과정을 볼 수 있다. 분류하는 기준은 알고리즘을 통해 자동으로 설정한다.

하지만 위와 같은 모델 구조는 너무 복잡해서 쉽게 파악하기 힘들다. 최대 깊이, 최소 분할 샘플 조건, 최소 리프 샘플 조건값을 설정해서 모델의 구조를 간단하게 해본다. 전문적으로 말하면 트리 모델의 하이퍼 파라미터를 튜닝한다.

```
# 결정 트리 모델 하이퍼 파라미터
# max_depth : 최대 깊이 조건 : 루트노드를 제외한 깊이 설정
# min_samples_split : 최소 분할 샘플 조건 : 각 노드에서 리프를 분할가능한 최소 값을 설정
# min_samples_leaf = 최소 리프 샘플 조건 : 마지막 리프의 최소 샘플 값을 설정
def iris_tree_hyper(max_dep = 4, min_ss = 4, min_sl = 2):
    dt_clf = DecisionTreeClassifier(random_state = 22,
                                    max_depth = max_dep,
                                    min_samples_split = min_ss,
```

```
                                    min_samples_leaf = min_sl)
    dt_clf.fit(X_train, y_train)

    export_graphviz(dt_clf, out_file="tree.dot",
                class_names= iris_data.target_names, # class 변수의 값
                feature_names = iris_data.feature_names, # 컬럼 이름 지정
                impurity = True, # 지니 계수(불평등 지수) 출력
                filled= True) # 그림 색상 여부

    with open("tree.dot") as f: # 저장한 파일 읽어오기
        dot_graph = f.read()


    pred = dt_clf.predict(X_train)
    accuarcy = accuracy_score(y_train, pred)
    print("학습 데이터 검증 정확도 : ", accuarcy)

    pred = dt_clf.predict(X_test)
    accuracy = accuracy_score(y_test, pred)
    print("테스트 데이터 검증 정확도 : ", accuracy)

    return graphviz.Source(dot_graph)

iris_tree_hyper()
```

학습 데이터 검증 정확도 : 0.9666666666666667
테스트 데이터 검증 정확도 : 0.9333333333333333




코드 5 출력 결과

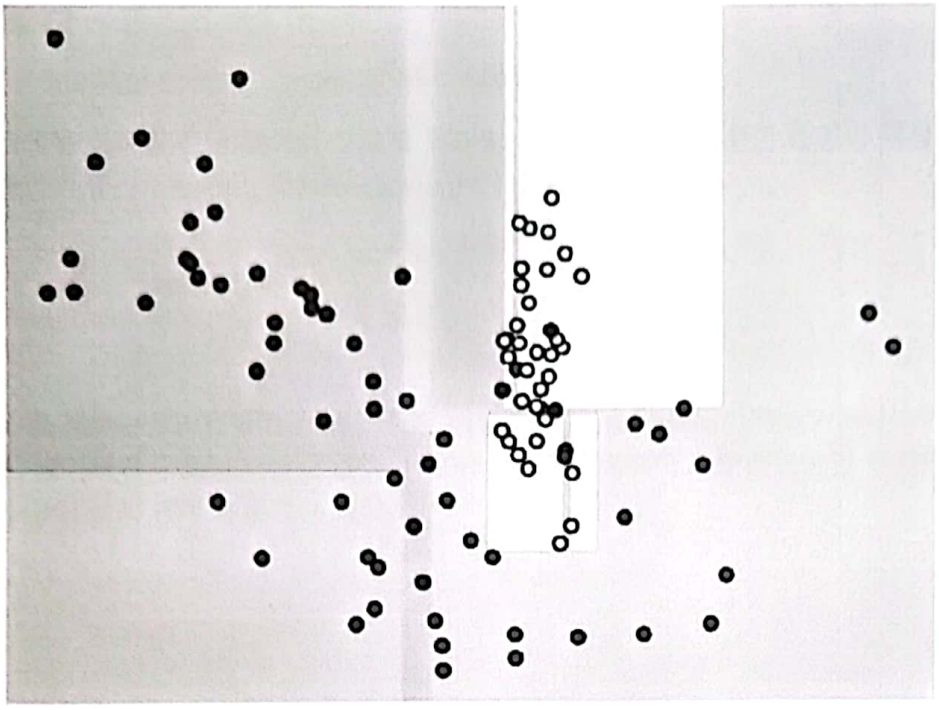
아까 아무런 설정을 하지 않은 모델보다 구조는 간단하지만, 학습데이터에 대해서는 정확도가 떨어지고, 테스트 데이터에 대해서는 정확도가 동일한것을 볼 수 있다.

그럼 모델을 복잡하게 하는것이 무조건 성능에 좋을까? 아래의 산점도를 보고 생각해보는 시간을 가져본다. X. 복잡할수록 train data에 대한 성능 ↑.

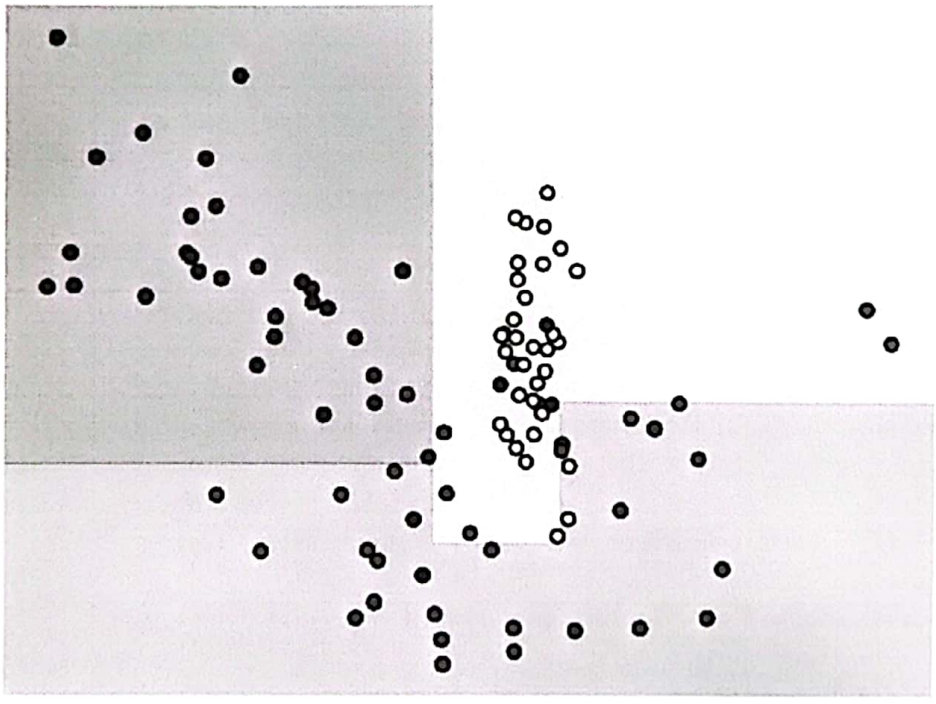
but 과적합 발생 → 실제 성능은 ↓.

물론, 단순하기만 해도 안 좋다.

↳ 과소적합 발생.



1) 트리 생성에 제약을 두지 않은 결정트리 모델



2) 트리 생성에 마지막 리프의 최소 크기는 6으로 설정한 결정트리 모델

# 개인 실습

데이터셋은 iris 대신 breast_cancer 데이터 셋을 사용한다.

이 데이터셋은 유방암 진단 데이터 셋으로 유방암 진단 사진으로 측정한 종양의 특징값을 사용하여 종양이 양성(benign), 악성(malignant)인지를 판단한다.

```
# breast_cancer 데이터셋을 불러온다.
from sklearn.datasets import load_breast_cancer
```

데이터의 총 개수는 569개이며, 테스트 데이터를 이전 실습과 동일하게 8:2로 나누어서 모델링한다. 데이터의 구조는 다음과 같다. (데이터 구조를 이해하고 모델링하면 좋지만, 실습 과정에서 굳이 보지 않아도 문제는 없다.)

```
**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and
the class

    :Attribute Information:
        - radius (mean of distances from center to points on the
perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the conto
ur)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of
the three
        largest values) of these features were computed for each
image,
        resulting in 30 features.  For instance, field 3 is Mean
Radius, field
        13 is Radius SE, field 23 is Worst Radius.
```

```
    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

===================================== ====== ======
                                       Min    Max
===================================== ====== ======
radius (mean):                        6.981  28.11
texture (mean):                       9.71   39.28
perimeter (mean):                     43.79  188.5
area (mean):                          143.5  2501.0
smoothness (mean):                    0.053  0.163
compactness (mean):                   0.019  0.345
concavity (mean):                     0.0    0.427
concave points (mean):                0.0    0.201
symmetry (mean):                      0.106  0.304
fractal dimension (mean):             0.05   0.097
radius (standard error):              0.112  2.873
texture (standard error):             0.36   4.885
perimeter (standard error):           0.757  21.98
area (standard error):                6.802  542.2
smoothness (standard error):          0.002  0.031
compactness (standard error):         0.002  0.135
concavity (standard error):           0.0    0.396
concave points (standard error):      0.0    0.053
symmetry (standard error):            0.008  0.079
fractal dimension (standard error):   0.001  0.03
radius (worst):                       7.93   36.04
texture (worst):                      12.02  49.54
perimeter (worst):                    50.41  251.2
area (worst):                         185.2  4254.0
smoothness (worst):                   0.071  0.223
compactness (worst):                  0.027  1.058
concavity (worst):                    0.0    1.252
concave points (worst):               0.0    0.291
symmetry (worst):                     0.156  0.664
fractal dimension (worst):            0.055  0.208
===================================== ====== ======

:Missing Attribute Values: None
```

```
:Class Distribution: 212 - Malignant, 357 - Benign
```

나머지 과정은 기존 실습과 코드는 동일하게 한다. 하지만, 모델 하이퍼 파라미터 중 min_samples_split, min_samples_leaf는 별도의 값을 설정하지 않고, 최대 깊이 조건만 2부터 10까지 설정해본다. 이 때 각 결과마다 학습 데이터 정확도와 테스트 데이터 정확도를 기록하고 총 결과에 대해서 왜 이런 결과가 나왔는지 분석해본다.