

데이터 사이언스

Decision Tree 구현.

2015004475 김태훈

1. 구동 환경

OS: Windows 10 64비트

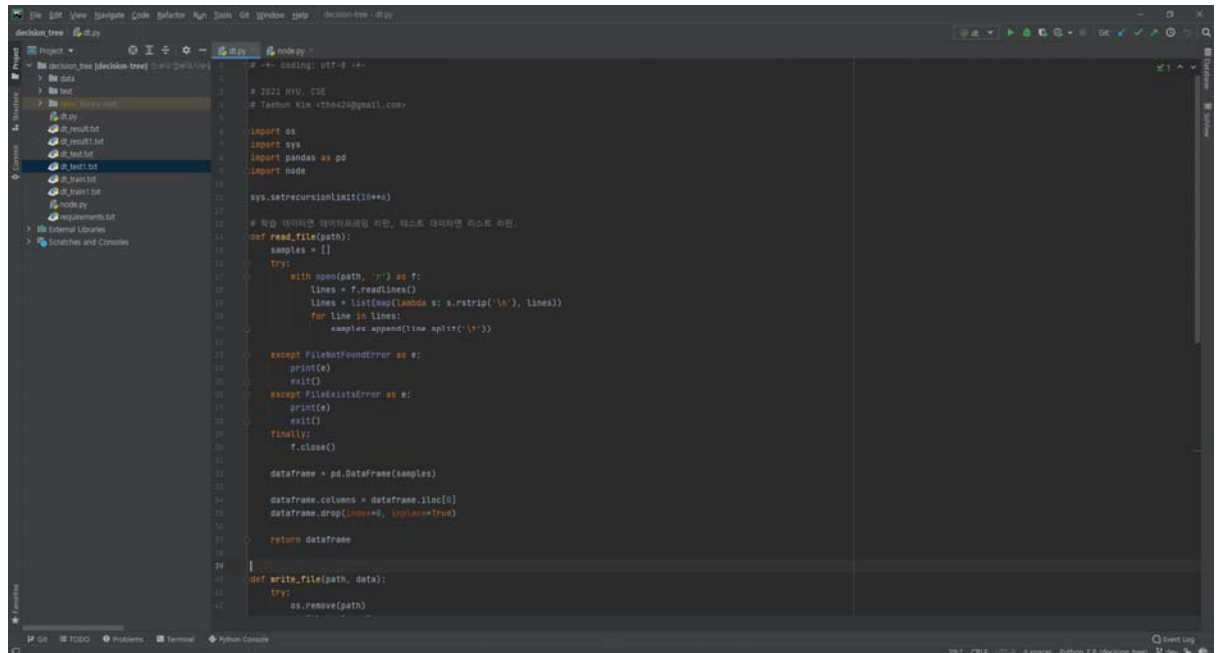
Language: Python 3.8

2. 프로그램 구조

1. 트레이닝 데이터, 테스트 데이터를 읽어 pandas Dataframe 으로 변환.
2. 트레이닝 데이터를 Node 객체에 전달하여 트리 노드 생성.
3. 트리 노드는 attribute를 설정하여 (gain ratio이용) 데이터를 분할, 재귀적으로 노드 생성
4. 마지막 leaf 노드는 가진 데이터셋의 클래스 라벨을 majority voting을 통해, 혹은 한가지 라벨 뿐인 경우 해당 라벨로 자신의 class_label을 정의함.
 - A. leaf가 아닌 경우라도 자식 노드 중 테스트 데이터 튜플의 라벨에 매치되지 않는 KeyError가 발생할 수 있기 때문에 일단 가진 데이터셋으로 majority voting을 실시하여 class_label을 보유함.
5. 트레이닝이 끝나면 테스트 데이터를 한 행(row)씩 읽어 트리에 적용.
6. 결과를 취합하여 result 파일에 작성.

3. 코드 설명

dt.py

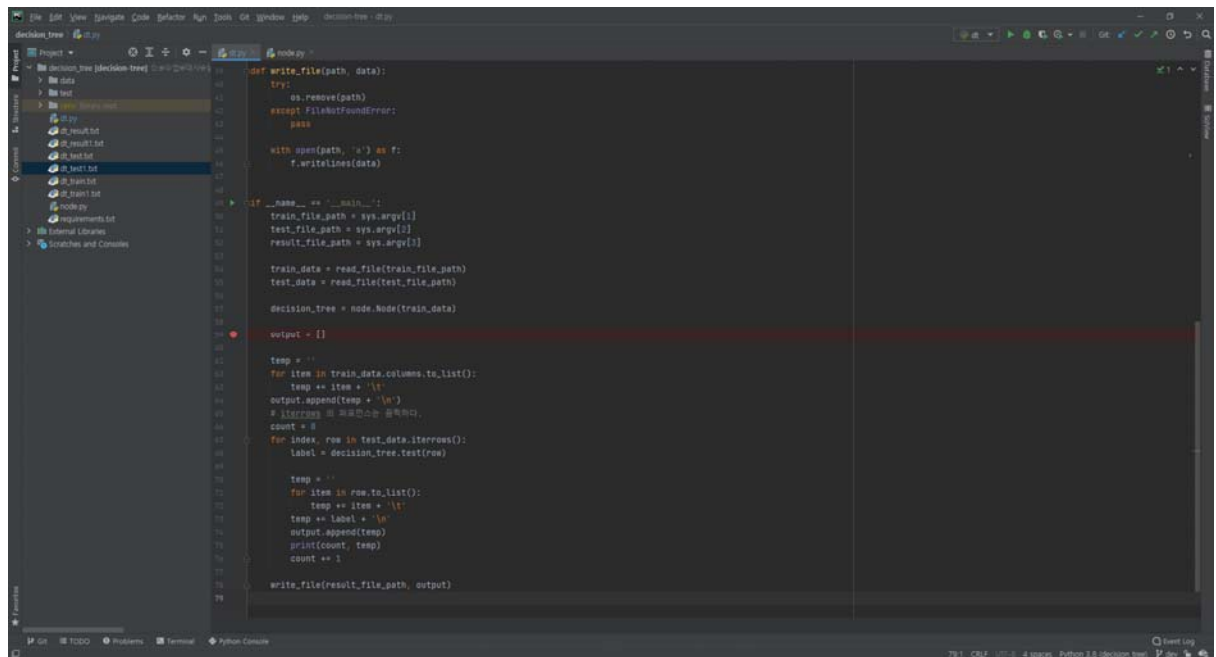


```
1 # -*- coding: utf-8 -*-
2
3 # 2021 HYU, CHS
4 # Taehun Kim <th42@gmail.com>
5
6 import os
7 import sys
8 import pandas as pd
9 import node
10
11 sys.setrecursionlimit(10**6)
12
13 # TODO: 재귀적인 데이터 처리를 위한, 텍스트 데이터의 리소스 처리.
14
15 def read_file(path):
16     samples = []
17     try:
18         with open(path, 'r') as f:
19             lines = f.readlines()
20             lines = list(map(lambda s: s.rstrip('\n'), lines))
21             for line in lines:
22                 samples.append(line.split('\t'))
23     except FileNotFoundError as e:
24         print(e)
25         exit()
26     except FileExistsError as e:
27         print(e)
28         exit()
29     finally:
30         f.close()
31
32     dataframe = pd.DataFrame(samples)
33
34     dataframe.columns = dataframe.iloc[0]
35     dataframe.drop(index=0, inplace=True)
36
37     return dataframe
38
39
40 def write_file(path, data):
41     try:
42         os.remove(path)
```

재귀 리미트를 10만으로 설정.

read_file

텍스트 파일을 읽어 pandas DataFrame 객체를 반환. 텍스트 파일 첫 줄은 컬럼 이름들이므로 적용하고 한 줄 삭제.



write_file

이미 파일이 존재하면 제거하고 새로 작성. data를 받아 path에 작성.

main

argv 받아 각각 파일 경로 저장. 파일로부터 데이터프레임 읽어들이어 트레이닝 데이터로 트리 객체 생성.

트리가 생성된 후 컬럼 이름을 output 리스트에 저장.

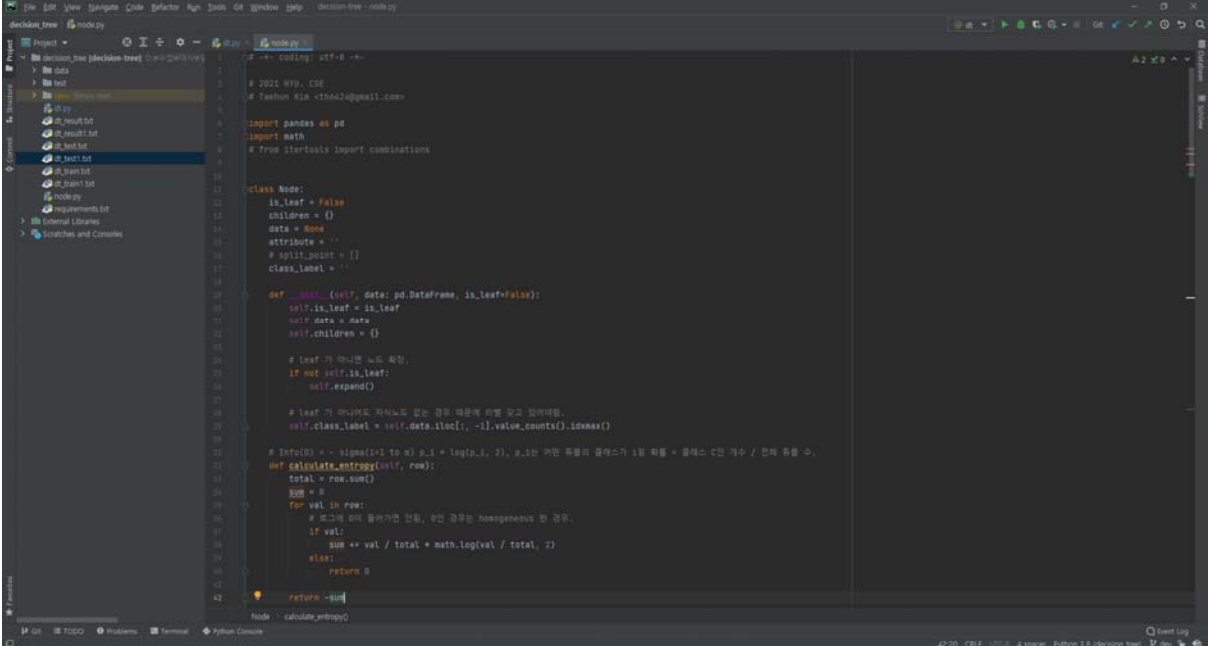
테스트 데이터프레임을 iterrows(index, row를 반환) 통해 한 튜플(샘플)씩 트리에 적용하여 테스트.

iterrows는 데이터프레임에서 최대한 지양해야 하나 본 과제를 수행하는데 큰 이슈가 될 정도는 아니라고 판단하여 사용.

트리에 적용 후 classification 결과를 받아 output에 저장.

루프가 끝나면 output 데이터를 파일에 작성.

node.py



```
# coding: utf-8 -*-
# 2021 WYB, CSE
# Tashun Kim <tkim24@gmail.com>

import pandas as pd
import math

# from itertools import combinations

class Node:
    is_leaf = False
    children = {}
    data = None
    attribute = ''
    split_point = {}
    class_label = ''

    def __init__(self, data: pd.DataFrame, is_leaf=False):
        self.is_leaf = is_leaf
        self.data = data
        self.children = {}

        # leaf 가 아닌 노드 확인.
        if not self.is_leaf:
            self.expand()

        # leaf 가 아닌 노드 처리하는 경우.
        self.class_label = self.data.iloc[:, -1].value_counts().idxmax()

    # Info(I) = - \sum_{i=1}^n p_i \log(p_i, 2), p_i는 어떤 클래스의 클래스가 i일 확률. 클래스 k는 개수 / 전체 개수.
    def calculate_entropy(self, row):
        total = row.sum()
        sum = 0
        for val in row:
            # 보기에 따라 다를 수 있음. 모든 경우는 homogeneous 한 경우.
            if val:
                sum += val / total * math.log(val / total, 2)
        else:
            return 0

    return -sum
```

Class Node

트리를 이루는 노드 객체. 노드가 꼬리를 물고 이어져 트리를 형성하는 구조.

bool is_leaf: 노드가 트리의 leaf라면 True, 쪼개는 분기(attribute 노드) 라면 False.

dict children: 노드의 자식 노드를 {label: Node} 형태로 저장. label은 해당 노드의 attribute에 속하는 라벨을 의미함. ex) age 를 attribute로 정했다면, {'>=30': node, '31...40': node, ...} 와 같은 children dict를 갖게 됨.

pandas.DataFrame data: 트레이닝 데이터 셋을 pandas Dataframe 객체로 받아 저장.

class_label: 본래 leaf 노드가 classification 결과를 반환할 수 있도록 하는 데이터였으나 테스트 데이터가 자신의 label에 맞는 노드를 찾지 못하는 경우 KeyError가 발생하여, 해당 에러 발생 시 leaf가 아니더라도 자신에게 남아있는 데이터를 majority voting을 통해 class_label을 반환할 수 있도록 모든 노드가 class_label을 하나씩 갖도록 함.

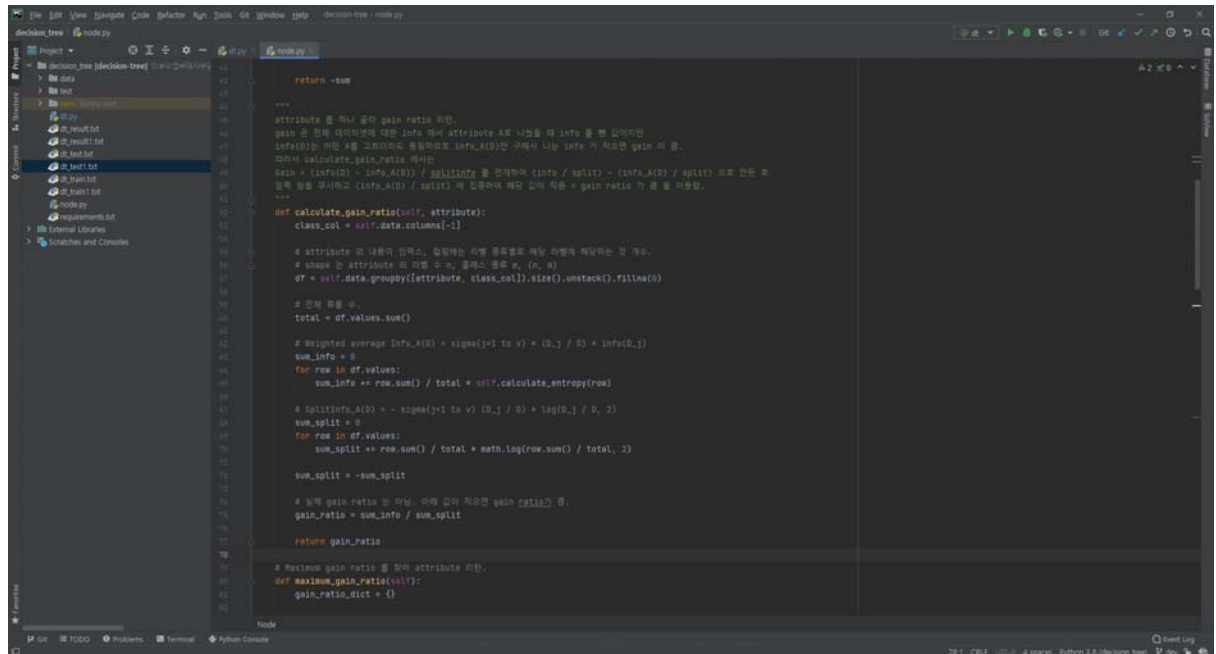
__init__

객체 생성자. data와 is_leaf를 받아 초기화. children 딕셔너리는 빈 딕셔너리로 초기화.

leaf 노드가 아니라면 트리를 확장. 이후 class_label을 지정함.

calculate_entropy

Information gain에서 entropy를 계산하는 수식을 구현. val 이 없는 케이스는 partition이 homogeneous 한 경우이기 때문에 값이 없는 것이 아니라 0이 들어있다고 간주하고 결과도 0이기 때문에 0을 리턴.



```
def calculate_gain_ratio(self, attribute):
    """
    attribute 를 하나 골라 gain_ratio 리턴.
    gain 은 전체 데이터셋에 대한 info 에서 attribute A로 나뉘었을 때 info 를 뺀 값이지만
    info(D)는 어떤 A를 고르더라도 동일하므로 info_A(D)만 구해서 나눈 info 가 작으면 gain 이
    큼. 따라서 calculate_gain_ratio 에서는 Gain = (info(D) - info_A(D)) / splitinfo 를 전개하여
    (info / split) - (info_A(D) / split) 으로 만든 후 앞쪽 항을 무시하고 (info_A(D) / split) 에 집중
    하여 해당 값이 작음 = gain ratio 가 큼을 이용함.
    """
    class_col = self.data.columns[-1]

    # attribute 의 내용이 인덱스, 값이라는 것을 염두해서 처리 수행해 주는 것 같음.
    # shape 는 attribute 의 인덱스 수, 클래스 종류 = (n, k)
    df = self.data.groupby([attribute, class_col]).size().unstack().fillna(0)

    # 전체 row 수.
    total = df.values.sum()

    # Weighted average Info_A(D) = - sum(p * log(p) / 2)
    sum_info = 0
    for row in df.values:
        sum_info += row.sum() / total * self.calculate_entropy(row)

    # SplitInfo_A(D) = - sum(p * log(p) / 2)
    sum_split = 0
    for row in df.values:
        sum_split += row.sum() / total * math.log(row.sum() / total, 2)

    sum_split = -sum_split

    # 실제 gain ratio 는 info, info_A(D) 가 작으면 gain ratio가 큼.
    gain_ratio = sum_info / sum_split

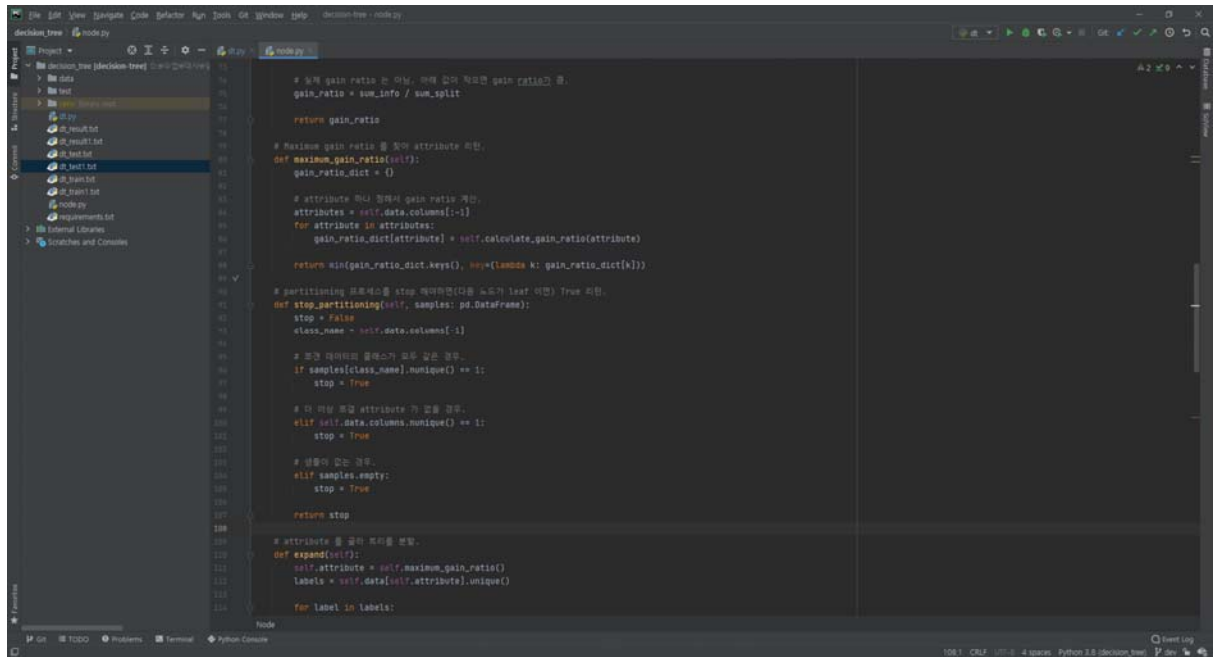
    return gain_ratio

# Maximum gain ratio 를 찾아 attribute 리턴.
def maximum_gain_ratio(self):
    gain_ratio_dict = {}
```

calculate_gain_ratio

attribute 를 하나 골라 gain ratio 리턴.

gain 은 전체 데이터셋에 대한 info 에서 attribute A로 나뉘었을 때 info 를 뺀 값이지만 info(D)는 어떤 A를 고르더라도 동일하므로 info_A(D)만 구해서 나눈 info 가 작으면 gain 이 큼. 따라서 calculate_gain_ratio 에서는 $\text{Gain} = (\text{info}(D) - \text{info}_A(D)) / \text{splitinfo}$ 를 전개하여 $(\text{info} / \text{split}) - (\text{info}_A(D) / \text{split})$ 으로 만든 후 앞쪽 항을 무시하고 $(\text{info}_A(D) / \text{split})$ 에 집중하여 해당 값이 작음 = gain ratio 가 큼을 이용함.

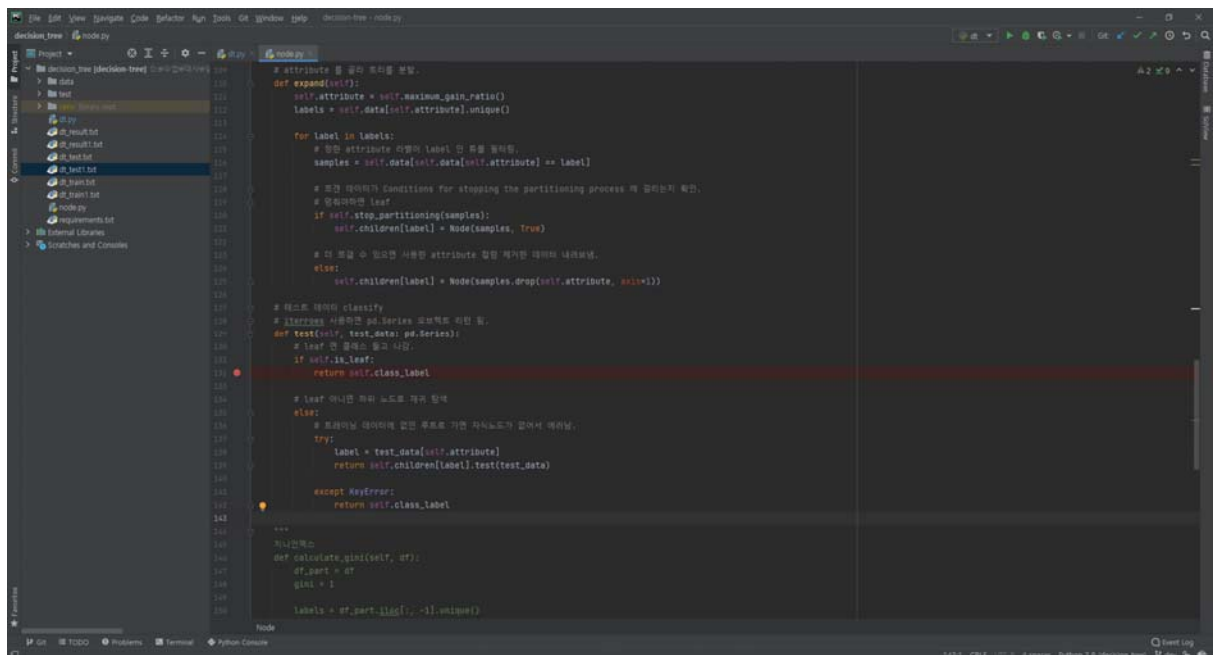


maximum_gain_ratio

각 attribute들의 gain ratio를 구해 가장 큰 gain ratio를 가지는 attribute를 선정, 반환.

stop_partitioning

Conditions for stopping the partitioning process 에 해당하는 3가지 경우를 구현.



expand

트리 확장. maximum gain ratio를 구해 attribute를 받아 지정하고, 해당 attribute의 label을 받아 각 label로 필터링 하여 partitioning이 필요하다면 추가 확장 가능한 노드를 생성, 아니면 leaf 노드를 생성하여 children dict에 추가.

test

테스트 데이터를 받았을 때 해당 노드가 leaf면 class label을 반환하고 하위 노드가 있다면 재귀적으로 탐색한다. 만약 children 딕셔너리에 테스트 데이터가 내려갈 수 있는 label이 없다면 dict가 KeyError를 발생시키고, 이 때는 학습되지 않은 경우이므로 <https://piazza.com/class/kltgwsjht2r236?cid=41> 을 적용해 남은 데이터로 majority voting 된 해당 노드 자체적 label을 리턴.

아래 주석 처리된 코드는 gini index 구현에 도전하다 실패한 코드.

4. 소스 컴파일 가이드

python 3.8로 작성된 스크립트이며, 추가로 필요한 패키지는 프로젝트 패키지에 포함된 requirements.txt에 기록되어 있음.