**Decision Tree Learning**

**for classification**

김상욱 교수님

2013002277 김재균

2017.04.18

**1. 개요**

의사결정나무(Decision Tree)는 의사 결정 지원 도구의 일종으로서 의사 결정 규칙과 그 결과들을 트리 구조로 도식화한 모델이다. 주어진 데이터를 분석하여 어떤 항목에 대해 관측값과 목표값을 연결시켜 예측 가능한 규칙들의 조합으로 나타낸다. 인공지능, 기계학습 그리고 통계분석 등 많은 분야에서 의사결정나무가 활용되고 있다.

**2. 목적**

이 프로젝트의 목적은 주어진 데이터(training data set)를 가지고 데이터 분석을 통해 트리 구조로 도식화한 예측 모델을 만든다. 그 후, 새로운 데이터(test data set)를 예측 모델을 통해 분류를 한다. 의사결정나무를 최적화하기 위해 가지치기 등 다른 알고리즘을 통해 성능을 개선해 나간다.

**3. 개발환경**

* Operating System: Windows 10 Pro
* Language: Python 3.6.4
* Source code editor: Visual Studio Code

**4. 프로그램 구조 및 알고리즘**

(1) 프로그램 실행 시 4가지 arguments(실행 파일, training data 파일, test data 파일, result 파일)를 입력한다. Training data 파일과 test data 파일에 있는 데이터를 읽어와 적절한 자료형에 저장한다.

(2) Training data의 가공이 끝난 후에는 탐욕 알고리즘(Greedy Algorithm) 기법으로 의사결정나무를 재귀하며 만든다.

A. Tree는 노드와 그에 따른 자식으로 객체화를 시킨 후, Gain Ratio를 통해 Root 노드의 첫번째 속성(Attribute)를 선정한다. 노드에는 속성이 저장되고, 속성의 라벨들이 노드의 자식이 된다.

B. 자식 라벨들을 기준으로 데이터를 분류하고 분류된 각각의 데이터 집합을 기 준으로 다시 Gain Ratio를 통해 다음 속성을 선정한다.

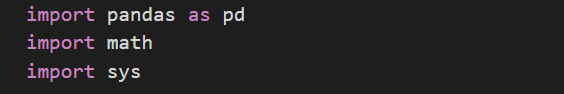
C. 선정된 속성을 노드에 저장하고 속성의 라벨들이 또 다시 노드의 자식이 된다.

D. 속성이 더 이상 없거나 Class Label의 종류가 하나일 때까지 B와 C의 방법을 재귀적으로 반복하며 트리를 만들어 나간다.

(3) 의사결정나무가 만들어 진 후, Test Data를 이 예측 모델에 적용 시킨다.

(4) 예측 모델의 결과를 바탕으로 의사결정나무 알고리즘을 더 최적화 하거나 더 이상의 Test Data가 없으면 종료한다.

**5. 코드 설명**

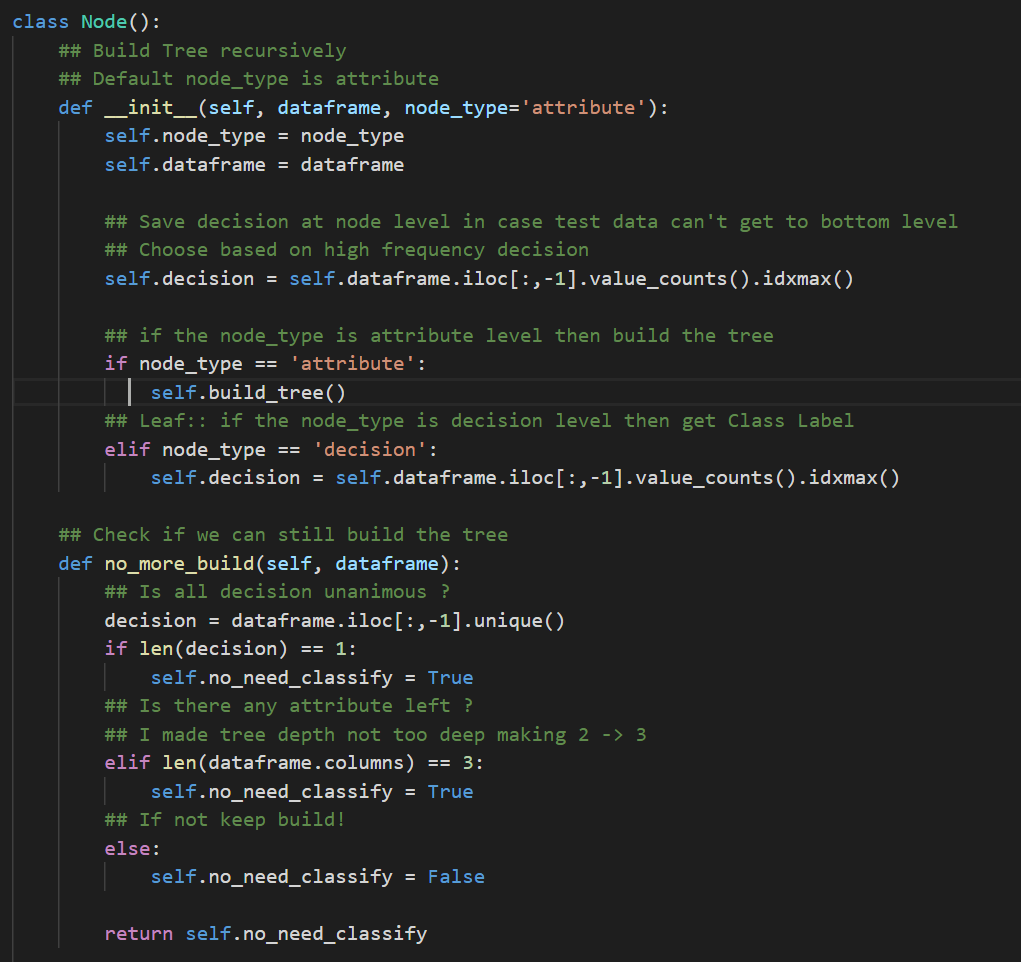


**사용한 모듈**

- pandas: 데이터프레임을 사용하기 위함이다.

- math: log 함수를 사용하기 위함이다.

- sys: argument를 불러오기 위함이다.



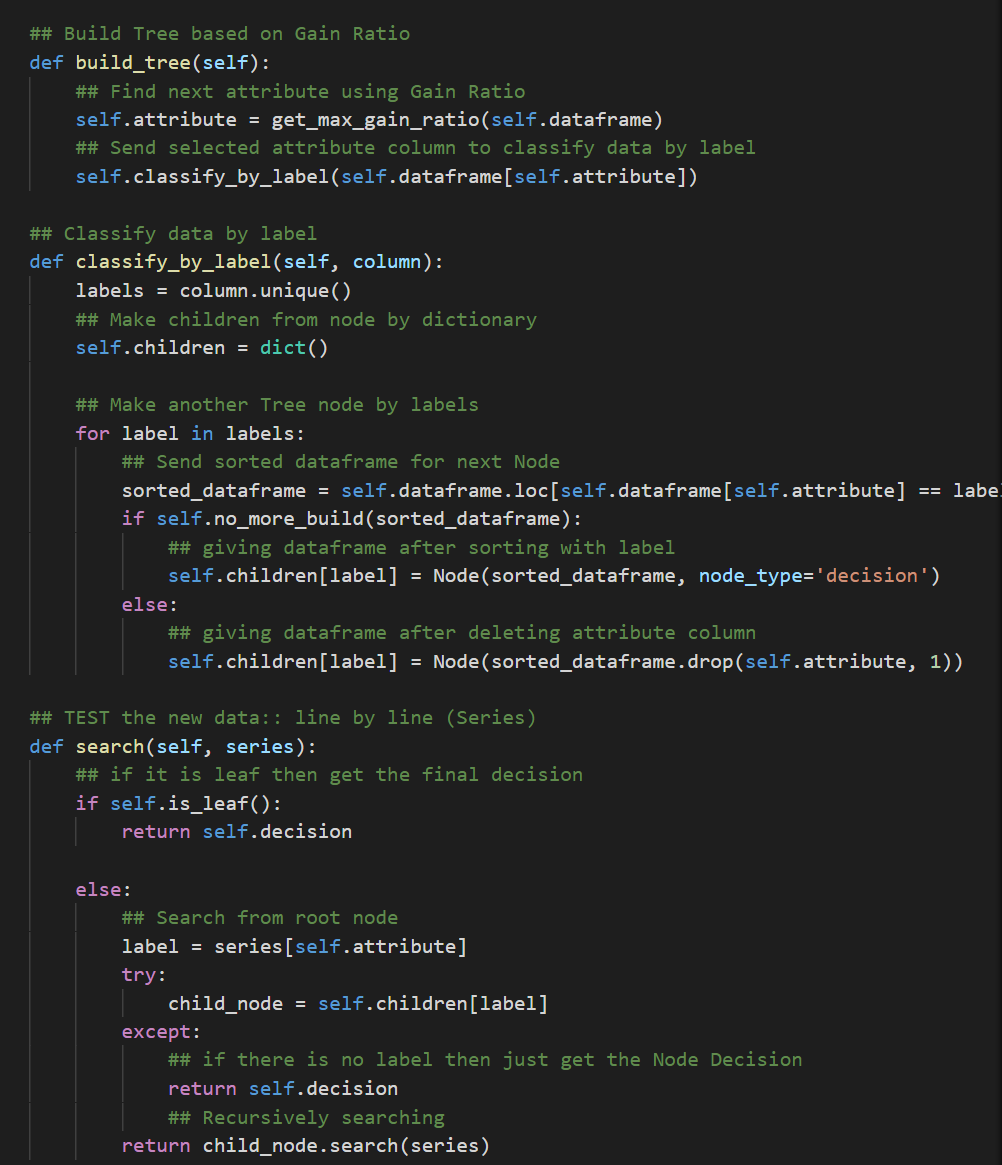
**Class Node()**

- 트리를 Class 함수를 사용하여 추상화 하였고, 노드는 속성을 저장하고 있고, 노드의 자식으로는 그 속성의 라벨들이 있다.

- node\_type은 속성을 저장하는 노드인지, 마지막 leaf 노드인지를 구분하기 위함이다.

- def \_\_init\_\_(): 노드를 만들 때는 node\_type이 default로는 attribute가 설정되고 각 노드마다 Class Label의 확률을 계산하여 decision 변수에 저장해 놓는다. (추후에 pruning을 하기 위함이다). Node\_type이 attribute라면 트리를 다시 만들고 decision이라면 leaf 노드이니 Class Label 중 가장 많이 Label을 decision에 저장한다.

- no\_more\_build(): 다음 단계에서도 트리를 만들 수 있는지의 여부를 확인해주는 함수이다. 만약에 모든 Class Label이 같은 카테고리라면 더 이상 attribute node를 만들 필요가 없다. 또한 속성이 더 이상 남지 않았다면 트리를 만들 필요가 없으며 더 이상 샘플이 없는 경우에도 마찬가지이다.

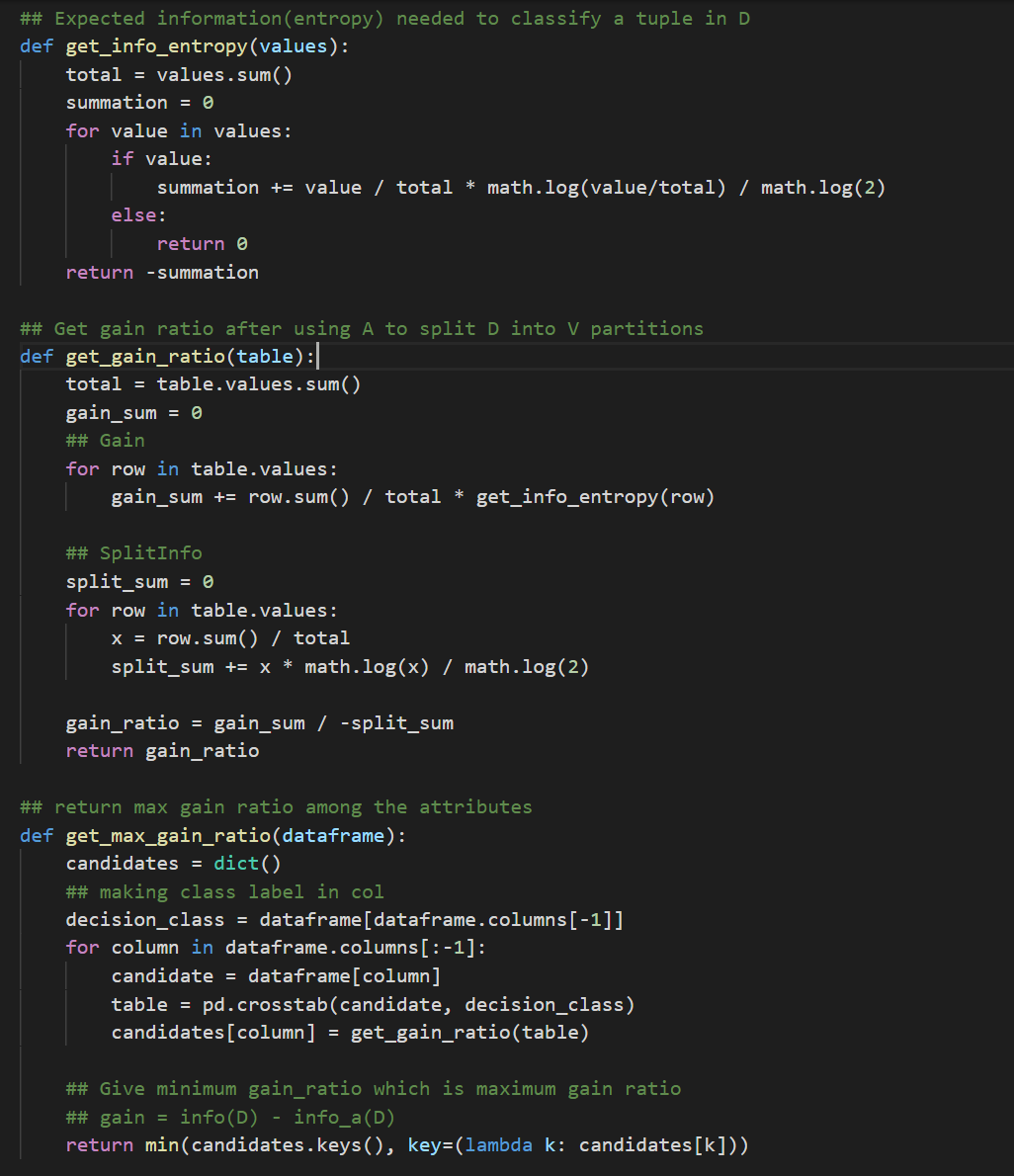


**Class Node()**

Build\_tree(): 트리를 만드는 함수이다. 먼저 Gain Ratio 함수를 통해 다음 노드의 attribute를 구하고 구한 attribute를 기준으로 분류하기 위해 classify\_by\_label() 함수에 넣어준다.

Classify\_by\_label(): 선택된 속성의 라벨들을 기준으로 분류하는 함수이다. no\_more\_build() 함수를 통해 더 이상 트리를 만들지 못하면 Node를 만들되, node\_type을 decision 넘겨준다. 트리를 다음 단계에서도 만들 수 있다면 라벨을 기준으로 분류된 데이터프레임을 Node 함수에 넘겨준다.

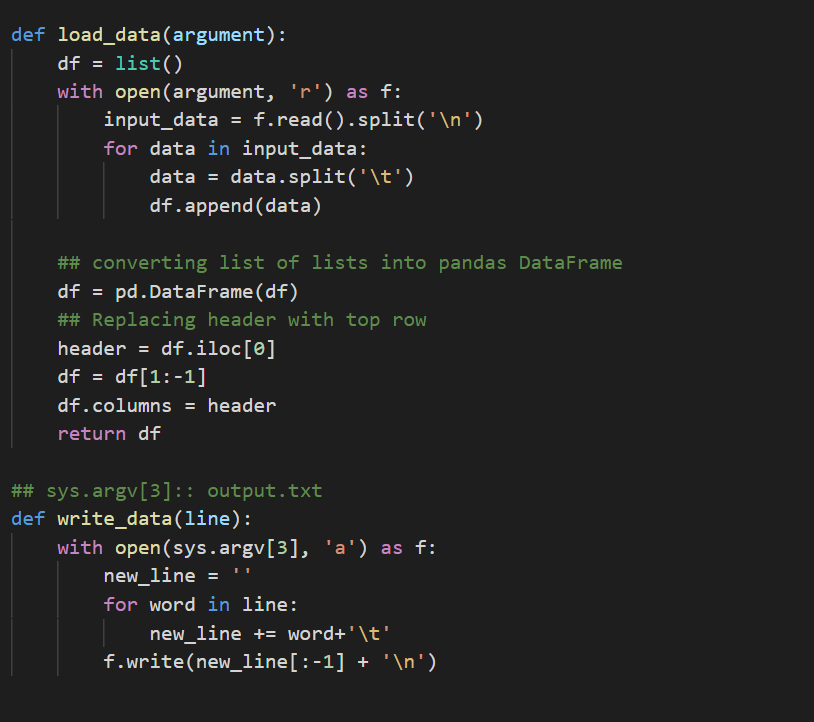
Search(): Test data를 적용하는 함수이다. 각 Test data의 아이템들을 Series 형태의 인자로 받아서 Training data로 만들어진 예측 모델에 재귀적으로 순회하며 Class label을 받는다.



Get\_info\_entropy(): Expected Information을 구하는 함수이다.

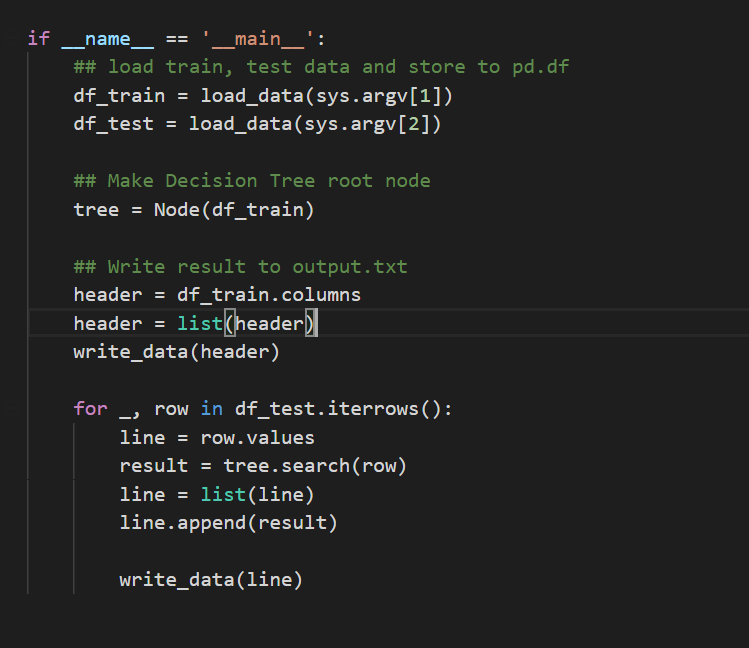
Get\_gain\_ratio(): get\_info\_entropy() 함수를 이용하여 본래 D 데이터 집합의 Expected information에서 A의 속성에 의해 분류한 후의 Expected information의 차이인 gain을 구한다. Information gain의 단점을 보완하기 위해 Split Information도 함께 고려하여 Gain Ratio를 구한다.

Get\_max\_gain\_ratio(): 속성들마다 구해진 Gain Ratio를 비교하여 가장 큰 Gain Ratio를 뽑아낸다. 여기서는 기존 D 데이터 집합의 Info 값이 같기 때문에 A의 속성에 의해 분류한 후의 Info 값이 가장 작은 속성을 도출해냈다.



Load\_data(): input.txt 파일로부터 데이터를 불러와 데이터프레임에 저장하는 함수이다.

Wirte\_data(): 결과를 output.txt 파일로 저장하는 함수이다.



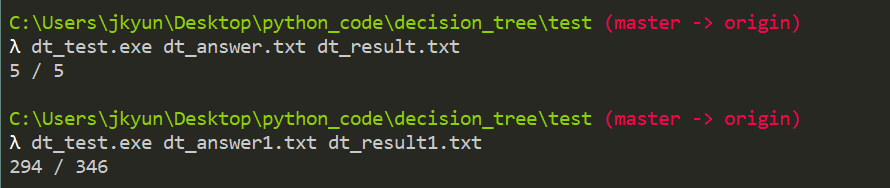
**Main 함수**

Load\_data()를 통해 Train data와 Test data를 불러와 데이터프레임에 저장하였다. Train data를 Node() 클래스 함수를 통해 tree(=의사결정나무)를 만들어주었다. 그 후, 결과를 output.txt 파일에 저장하기 위해 Test data와 도출된 결과를 합쳐주었다.

**6. 가이드 라인 및 실행 결과**

terminal(command) 명령어: python dt.py dt\_train.txt dt\_test.txt dt\_result.txt

**실행 결과**



**7. 에필로그**

이번 프로젝트를 통해 Classification의 대표 알고리즘, 의사결정나무(Decision Tree)를 배울 수 있었다. Tree를 추상화하여 재귀적으로 구현하는 것이 다소 어려움이 있었으나 그 과정에서 많은 것을 배울 수 있었다.

**8. 참조**

Data mining: concepts and techniques, Morgan Kaufmann  
https://en.wikipedia.org/wiki/decisiontree