

Data Science (ITE4005)

< Programming Assignment #2 >

Decision Tree – Gain Ratio

2016025732 이영석

2021-04-13

1. Summary

Gain Ratio 방식을 이용해 Decision Tree를 구축하고, Test Data의 Class Label을 Predict한다.

```
class Node():
    def __init__(self):
        self.criteria_attribute = None
        self.label_predict = None
        self.leaf = {} # key : attribute value, value: leaf node pointer
```

```
class DecisionTree():
    def __init__(self):
        self.root = Node()

        self.df_train = None
        self.df_test = None

        self.attribute = [] # (ex. age, income, etc..)
        self.label_name = None # (ex. buy_computer)
        self.label_value = [] # label_name으로 구분한 row (ex. no, yes)
```

Class는 Node, DecisionTree 두개로 구성된다.

Node의 label_predict는 해당 노드의 예측 label 변수가 저장된다. (by Majority Voting)

criteria_attribute는 Gain Ratio를 통해 얻어진 best_attribute 이름이 저장된다.

DecisionTree의 df_train/df_test는 각각 train data와 test data가 dataframe으로 저장된다.

```

if __name__ == '__main__':
    tree = DecisionTree()
    tree.set_data(sys.argv[1], sys.argv[2])
    tree.train_data()
    tree.test_data()
    tree.print_output(sys.argv[3])

```

주 알고리즘은 다음과 같은 순으로 구성하였다.

Setting Data -> Training Data -> Predicting Test Data's Label -> Printing Output

각 단계(모듈)는 다음과 같이 구성하였다. (자세한 내용은 2. Detailed Description 참고)

- A. set_data(arg1, arg2)
- B. train_data()
 - i. grow_tree(data)
 - 1. get_best_attribute(data)
 - A. info(data)
 - B. expected_info(data, attribute)
 - C. split_info(data, attribute)
- C. test_data()
 - i. predict(data)
- D. print_output(arg3)

2. Detailed Description

- A. set_data(arg1, arg2)
 - ➔ train 파일과 test 파일의 dataframe화 진행.
 - ➔ attribute, label_name, label_value 추출
- B. train_data()

i. `grow_tree(data)`

- ➔ majority voting한 label 해당 노드에 저장
- ➔ Best Attribute를 받아 Data를 나누고 해당 attribute는 버리며 self-recursive call하며 확장
- ➔ 루트 노드부터 차근차근 확장 (Top-Down / Divide & Conquer)
- ➔ 해당 노드와 다음노드만을 고려 (Greedy)

```
def grow_tree(self, node, data):  
  
    # Predict label by majority voting  
    label, counts = np.unique(data[self.label_name], return_counts=True)  
    node.label_predict = label[np.argmax(counts)]  
  
    if self.is_homogenous(data):  
        return  
  
    best_attribute = self.get_best_attribute(data)  
  
    # Set criteria attribute  
    node.criteria_attribute = best_attribute  
    attribute_value = np.unique(data[best_attribute])  
  
    # Recursively grow  
    for i in range(len(attribute_value)):  
        node.add(attribute_value[i], Node())  
        _data = data.get(data[best_attribute]==attribute_value[i]) #해당 attribute에서 같은 value만 가진 data select  
        _data = _data.drop(best_attribute, axis=1) #해당 attribute행 삭제  
        self.grow_tree(node.leaf[attribute_value[i]], _data)
```

ii. `get_best_attribute(data)`

- ➔ 모든 attribute별로 gain ratio를 구하고, 가장 큰 값(이전과 다음 노드 엔트로피 차이가 가장 큰 값)을 가진 Attribute return

```
def get_best_attribute(self, data):  
    temp = 0  
    best_attribute = None  
    temp_attribute = data.columns.tolist()  
    del temp_attribute[-1] #Class label은 제외  
  
    for i in range(len(temp_attribute)):  
        gain = self.info(data) - self.expected_info(data, temp_attribute[i])  
        gain_ratio = gain / self.split_info(data, temp_attribute[i])  
        if temp < gain_ratio:  
            temp = gain_ratio  
            best_attribute = temp_attribute[i]  
  
    return best_attribute
```

1. `info(data)`

➔ 해당 노드에 해당하는 Entrophy값 return

2. expected_info(data, attribute)

➔ test attribute로 내려간 자식 노드에 해당하는 Entrophy return

3. split_info(data, attribute)

➔ gain ratio = gain / split_info

➔ gain ratio를 구하기 위한 분모값 return

(나누는 이유: attribute value 개수 별로 가중치 주는 것을 없애기 위함)

C. test_data()

➔ test data 열마다 predict함수 실행

iii. predict(data)

➔ Tree의 leaf까지 이동후, 해당 노드의 label_predict 값 새 행에 저장.

➔ ('Door == 3' 같은 Training Data에 없는 내용이 들어올 경우 try catch로 예외처리. Dictionary의 key가 없으면 해당 노드의 label_predict를 return한다)

D. print_output(arg3)

➔ Class Label이 추가된 data frame을 txt 파일로 저장

3. Instruction for compiling

A. python 설치

B. train file, test file 같은 디렉토리 내 위치

- C. Execute the program with three arguments: training file name, test file name, output file name

```
python DecisionTree_2016025732.py dt_train1.txt dt_test1.txt dt_result1.txt
```

4. Testing

- A. 3번 과 같이 실행
- B. Result

321	vhigh	low	4	more	med	med	acc
322	high	high	4	more	small	low	unacc
323	med	low	3	more	small	med	acc
324	med	med	5more	4	big	low	unacc
325	med	high	3	4	small	med	unacc
326	med	low	2	2	med	med	unacc
327	med	med	2	4	big	med	acc
328	vhigh	med	2	more	med	low	unacc
329	high	high	5more	more	big	low	unacc
330	vhigh	med	4	4	small	high	acc
331	low	med	3	more	small	med	acc
332	vhigh	vhigh	3	4	small	med	unacc
333	low	low	3	4	med	high	vgood
334	high	high	3	2	small	high	unacc
335	med	high	5more	2	med	high	unacc
336	high	med	2	4	small	low	unacc
337	high	med	2	more	big	med	acc
338	low	med	3	2	med	med	unacc
339	low	low	4	more	med	high	vgood
340	high	vhigh	2	4	med	med	unacc
341	med	vhigh	2	2	small	med	unacc
342	high	high	4	4	med	high	acc
343	vhigh	low	4	4	small	high	acc
344	vhigh	low	3	4	big	low	unacc
345	high	high	3	4	big	med	acc
346	med	low	5more	4	big	med	good
347	vhigh	high	5more	more	med	med	unacc

- C. 채점 모듈 실행 결과

➔ Gain Ratio (제출본)

```
C:\Users\이영석\Desktop\test>dt_test.exe dt_answer1.txt dt_result1.txt
323 / 346
```

➔ Information Gain (대조군)

```
C:\Users\이영석\Desktop\test>dt_test.exe dt_answer1.txt result.txt  
320 / 346
```

근소한 차이로 Gain Ratio의 정확도가 더 높았다.

결론 : 이를통해 Information Gain의 단점(Attribute Value들의 개수가 많을수록 Best Attribute가 되기 쉬움)을 Gain Ratio가 보완한다는 것을 알 수 있었다.

다만, Gain Ratio의 단점은 unbalanced splits에 우위를 두는데, balanced splits에 우위를 두고 싶다면 Gini Index 방식을 쓰는 것이 효율적이다. Training Set의 특성에 맞춰 어떤 Decision Tree Algorithm을 선택할지가 관건이겠다.