

Decision Tree - 2016025950 정재용

알고리즘 요약

- 구현된 함수는 5개입니다.
 - `get_entropy()` - 엔트로피 값을 계산하는 함수
 - `get_info_gain()` - 애틀리뷰트의 information gain을 계산하는 함수
 - `get_next_attr()` - information gain을 비교하여 다음 기준 애틀리뷰트를 선정하는 함수
 - `decision_tree()` - 결정트리 모델을 만드는 함수
 - `test_decision_tree()` - 트리 모델을 이용해서 주어진 테스트 파일을 분류하는 함수
- 트리를 구현하는데는 dict type을 사용했습니다.
 - ex) {'decision_tree': {'age': {'≤30' : yes, '31...40' : yes, '>40' : no}}}
- 비교할 속성이 없는 경우 분류할 때 majority voting 정책을 따랐습니다.
- 분류할 때 tree에 없는 값을 만났을 때는 해당 노드의 class 중 다수결을 따랐고, 아직 결정되지 않은 노드라면 첫번째 하위 노드로 분류를 이어가게 했습니다.

구현 함수

`get_entropy()`

```
def get_entropy(data):  
    #  
    # 주어진 data의 entropy를 계산  
    # return (float): data's entropy  
    #  
    key, counts = np.unique(data, return_counts=True)  
    return -np.sum([counts[i] / np.sum(counts) * np.log2(counts[i] / np.sum(counts)) f  
    or i in range(len(key))])
```

- data는 class값으로 이루어진 series / array
- class value와 그 count를 가지고 $-\sum_{i=1}^m p_i \log_2(p_i)$ 수식을 계산

get_info_gain()

```
def get_info_gain(data, attr, target):
    #
    # attr의 information gain을 계산
    # data (DataFrame): 전체 data
    # attr (str): information gain을 계산할 attribute
    # target (str): 분류의 대상이 되는 class
    # return (float): information gain of attribute
    #

    # attribute 전체의 entropy
    total_entropy = get_entropy(data[target])

    # attribute 값들의 entropy를 구하고 가중 평균을 구한다.
    labels, counts = np.unique(data[attr], return_counts=True)
    attr_entropy = np.sum([cnt * (get_entropy(data[data[attr] == l][target])) for l, cnt in zip(labels, counts)]) / np.sum(counts)

    return total_entropy - attr_entropy
```

- attribute의 값들의 비율을 구한다.
- $Info_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} * Info(D_j)$ 수식을 계산해서 attribute entropy의 가중 평균을 구한다.
- 전체 entropy도 구한 후 information gain을 구해 리턴한다.

get_next_attr()

```
def get_next_attr(data, target):
    #
    # information gain을 비교해서 다음 기준 attribute를 선정
    # data (DataFrame): data for classification
    # return (str): next attribute name
    #

    attrs = list(data.columns[:-1])
    index = np.array([get_info_gain(data, attr, target) for attr in attrs]).argmax()

    return attrs[index]
```

- data의 모든 attribute에 대해 information gain을 구한다.
- 최대 information gain을 갖는 attribute를 구해서 그 이름을 리턴한다.

decision_tree()

```
def decision_tree(data, target):
    #
    # 주어진 data로부터 decision tree model을 생성
    # data (DataFrame): train data
    # target (str): 분류의 대상이 되는 class
    # return (dict): node of decision tree
    #

    # 남아있는 data들의 class 값이 일치할 때
    if len(np.unique(data[target])) <= 1:
        return np.unique(data[target])[0]

    # 비교할 속성이 없을 때
    elif len(data.columns) <= 1: # class column만 남음
        key, counts = np.unique(data[target], return_counts=True)
        return key[counts.argmax()] # majority voting 방식으로 class 값 리턴

    else:
        # 다음 attribute를 선정하고 dict type으로 node를 만든다
        next_attr = get_next_attr(data, target)
        node = {next_attr: {}}
        labels = np.unique(data[next_attr])
        # attribute의 값에 따라 tree 연장
        for l in labels:
            sub_data = data[data[next_attr] == l].drop([next_attr], axis=1)
            node[next_attr][l] = decision_tree(sub_data, target)

        return node
```

- 노드는 dict type
- 자식 노드는 함수를 재귀 호출해서 생성한다.
 - 재귀호출할 때 해당 노드로 분류된 튜플들을 전달한다.
 - 남은 튜플들의 class가 하나로 일치하거나, 더 이상 분류 기준 속성이 없을 때 재귀를 끝낸다.
 - 남은 속성이 없을 때는 남은 튜플들에 대해 majority voting으로 classify한다.

test_decision_tree()

```

def test_decision_tree(decision_tree, test_file, target):
    #
    # 주어진 test 파일을 decision tree model으로 분류
    # decision_tree (dict): tree model
    # test_file (DataFrame): classify test file
    # target (str): class column name (classification answer)
    # return (DataFrame): test file's classification answer
    #

    answer = []
    for i in range(len(test_file.index)):
        subtree = decision_tree
        # class value (str)을 만날 때까지 반복
        while type(subtree) == type({}):
            attr = list(subtree.keys())[0]
            label = test_file.loc[i, attr]

            if label in subtree[attr]: # 트리에 있는 값
                subtree = subtree[attr][label]
            else:
                # 트리에 없는 값을 만났을 때
                # class가 결정되는 attr과 노드가 이어지는 attr을 구한다.
                values = list(subtree[attr].values())
                target_value = [v for v in values if type(v) == type('')]
                next_node = [n for n in values if type(n) == type({})]

                if len(target_value) > 0: # class가 결정되는 값이 있다면 그 중 더 많은 값으로
결정
                    key, counts = np.unique(target_value, return_counts=True)
                    subtree = key[counts.argmax()]
                else: # 아니라면 다음 노드로
                    subtree = next_node[0]

            answer.append(subtree)

        test_file[target] = answer

    return test_file

```

- test file의 row tuple들을 순회하면서 classify 한다.
- 현재 노드가 dict type이라면 (분류 될 수 있다면) 튜플의 애트리뷰트 값을 트리와 비교 한다.
 - 트리에 있는 값을 가졌다면 트리를 따라 내려간다.
 - 트리에 없는 값을 가졌다면 분류가 결정되는 하위 노드와 분류가 계속되는 하위 노드를 구분한다.
 - 분류가 결정될 수 있다면 결정되는 class들 중 다수결을 따라서 결정한다.
 - 분류가 결정될 수 없다면 첫번째 하위 노드로 내려가서 분류를 이어간다.

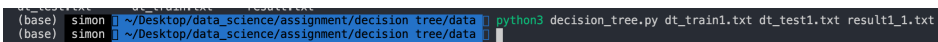
- 완성된 answer list를 test file의 column으로 더하고 리턴한다.

main

```
if __name__ == '__main__':  
    # command line arguments  
    train_file_name = sys.argv[1]  
    test_file_name = sys.argv[2]  
    result_file_name = sys.argv[3]  
  
    # open files  
    train_file = pd.read_csv(train_file_name, sep='\t')  
    test_file = pd.read_csv(test_file_name, sep='\t')  
    target = train_file.columns[-1]  
  
    # generate tree model & test  
    decision_tree = decision_tree(train_file, target)  
    result = test_decision_tree(decision_tree, test_file, target)  
  
    result.to_csv(result_file_name, sep='\t', index=False)
```

- train_file로 트리 모델을 만들고, test함수로 결과 df를 얻는다.

실행



```
(base) simon [~/Desktop/data_science/assignment/decision tree/data] python3 decision_tree.py dt_train1.txt dt_test1.txt result1_1.txt  
(base) simon [~/Desktop/data_science/assignment/decision tree/data]
```

result1.txt						
buying	maint	doors	persons	lug_boot	safety	car_evaluation
med	vhhigh	2	4	med	med	acc
low	high	4	4	small	low	unacc
high	vhhigh	4	4	med	med	acc
high	vhhigh	4	more	big	low	unacc
low	high	3	more	med	low	unacc
med	high	2	more	small	high	acc
vhhigh	low	3	2	med	high	unacc
med	high	2	4	small	low	unacc
med	low	5more	4	small	med	good
med	low	5more	2	big	med	unacc
med	low	4	more	big	high	vgood
low	low	4	2	big	high	unacc
low	low	3	more	med	low	unacc
high	med	2	2	big	high	unacc
high	low	4	more	small	low	unacc
med	vhhigh	3	4	med	med	acc
low	low	3	more	small	high	good
vhhigh	med	2	more	med	med	acc
vhhigh	low	4	more	big	high	acc
vhhigh	low	2	2	small	high	unacc
high	high	5more	2	big	med	unacc
high	med	5more	2	med	low	unacc
med	med	3	2	big	high	unacc
low	vhhigh	5more	2	small	low	unacc
vhhigh	vhhigh	3	more	small	high	unacc
low	high	2	more	big	med	acc
vhhigh	vhhigh	5more	more	small	high	unacc
high	low	5more	4	small	med	unacc
low	med	4	2	big	low	unacc
med	vhhigh	3	2	small	med	unacc
med	high	4	more	small	med	unacc
vhhigh	low	4	4	big	low	unacc
high	low	5more	4	big	med	acc
low	vhhigh	3	more	small	med	unacc
vhhigh	low	5more	2	small	low	unacc
low	vhhigh	2	2	med	med	unacc
med	med	4	4	big	low	unacc
med	high	4	more	big	high	acc
high	high	5more	4	big	high	acc
med	vhhigh	4	more	med	low	unacc
low	med	3	4	small	high	good
vhhigh	med	3	2	med	low	unacc

C:\Users\kolok\Desktop\4학년 1학기\데이터 사이언스\실습\assignment 2\test>dt_test.exe dt_answer1.txt result1_1.txt
320 / 346