Programming Assignment #2

Index

학번: 2017041930

이름: 김선동

1. Environment & Requirements

OS는 Windows, language는 python을 사용하였다. Editor는 visual studio code를 사용하였다. 사용한 python의 버전은 3.10.4이다.
별도의 패키지를 사용하였는데 requirements.txt를 첨부하였으니 아래 명령어로 설치하면 된다.

> pip3 install -r requirements.txt

2. How to run?

> ./dt.py dt_train.txt dt_test.txt dt_result.txt

위 명령어를 실행하면 training set으로 만든 모델로 test data로 분류한 결과가 dt_result.txt에 저장된다.

3. Summary of my implementation

```
# ./dt.py dt_train.txt dt_test.txt dt_result.txt
# Attribute_selection_method: information gain
if __name__ == '__main__':
    if len(sys.argv) != 4:
        print("Execute the program with tree arguments: training file name, test file name, output file name")
        print("Training file name = 'dt_train.txt', test file name = 'dt_test.txt', output file name = 'dt_result.txt'")

    training_file_name = sys.argv[1]
    test_file_name = sys.argv[2]
    output_file_name = sys.argv[3]

    training_set = pd.read_csv(training_file_name, sep="\t")
    test_set = pd.read_csv(test_file_name, sep="\t")
    class_label = training_set.columns[-1] # Class:buys_computer

    decision_tree = Generate_decision_tree(training_set)
    Classifier(decision_tree, test_set, class_label).to_csv(output_file_name, sep='\t', index=False)
```

정해지지 않은 n개의 컬럼에 대해 프로그램이 동작해야 하므로, pandas를 사용하였다. Read_csv함수를 사용하면 위와 같은 동작이 가능하다. 그렇게 입력처리를 해준다. 이후 Generate_decision_tree함수에 training set을 인자로 넘겨주어 decision tree를 만들어준다. 이 때 attribute selection method는 information gain을 사용했다. 얻은 decision tree와 test set, class label을 인자로 주어 분류를 진행하고 이를 dt_result.txt에 저장하는 것이 Classifier함수이다. 이제 함수 하나씩 살펴보자.

```python
def Generate_decision_tree(training_set):
    # Training data로부터 entropy 계산
    def Get_entropy(training_tuples):
        attribute, counts = np.unique(training_tuples, return_counts=True)
        entropy = float(0)
        for i in range(len(attribute)):
            p_i = counts[i]/np.sum(counts)
            entropy -= p_i * np.log2(counts[i]/np.sum(counts))
        return entropy

    # Information gain = entropy(parent) - [weighted average]entropy(children)
    def Get_information_gain(training_tuples, attribute_name, target_attribute):
        parent_entropy = Get_entropy(training_tuples[target_attribute])
        attributes, counts = np.unique(training_tuples[attribute_name], return_counts=True)
        children_entropy = float(0)
        for attribute, count in zip(attributes, counts):
            children_entropy += np.sum([count * (Get_entropy(training_tuples[training_tuples[attribute_name] == attribute][target_attribute]))]) / np.sum(counts)
        return parent_entropy - children_entropy

    def Get_best_attribute(training_tuples, target_attribute):
        attribute_list = list(training_tuples.columns[:-1])
        best_information_gain = float(0)
        best_attribute = attribute_list[0]

        for attribute in attribute_list:
            cur_information_gain = Get_information_gain(training_tuples, attribute, target_attribute)
            if(best_information_gain < cur_information_gain):
                best_attribute = attribute
                best_information_gain = cur_information_gain

        return best_attribute
```

Training data를 인자로 받아 decision tree를 생성하는 함수이다. 우선 내가 채택한 attribute selection method는 information gain이므로 이와 관련된 함수 두 개를 만들었다. 먼저 엔트로피를 계산하는 Get_entropy 함수이다. 엔트로피를 계산하는 공식은 아래와 같다.

$$Entropy = H(S) = \sum_{i=1}^{c} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{c} p_i \log_2 p_i$$

결국 어떤 사건 i가 발생할 확률에 그 사건이 갖는 정보량을 곱한 것들의 합이다. 위 식에 따라 entropy를 계산해주고 이 변화량을 나타내는 information gain을 구해준다. Information gain을 구하는 공식은 다음과 같다.

$$\begin{aligned} Information\ Gain = IG(S, A) &= H(S) - H(S, A) \\ &= H(S) - \sum p(t) H(t) \end{aligned}$$

곧, 분기 전후의 엔트로피의 차이를 구하는 것이다. 전을 parent라고 하고, 후를 children라고 한다면 children에 양쪽 가지로 나뉘는 확률 p(t_i)를 가중치로 곱한 후 합쳐진다.

마지막으로, Get_best_attribute는 분기를 할 때 해당 attribute의 information gain이 제일 높은 값을 우선해서 택해주어야 한다. (More homogeneous하기 때문) 그러기 위해서 각 attribute마다 information gain을 구해주고 제일 큰 값을 택해준

다. 만약 다 똑같다면 첫 번째 attribute를 골라준다.

```
# create root node for the tree
decision_tree = {}
class_label = training_set.columns[-1] # Class:buys_computer

# if all members of training set are in the same class C
if len(np.unique(training_set[class_label])) <= 1:
    return np.unique(training_set[class_label])[0]

# if Attributes are empty then single-node tree with label
# most common value of Target_attribute in training_set
elif len(training_set.columns) <= 1:
    key, counts = np.unique(training_set[class_label], return_counts=True)
    return key[counts.argmax()]

else:
    # Get Best attribute(decision attribute)
    best_attribute = Get_best_attribute(training_set, class_label)
    decision_tree[best_attribute] = {}

    # Get best attribute's values
    best_attribute_values = np.unique(training_set[best_attribute])

    # For each possible value of best attribute's values
    for attribute_value in best_attribute_values:
        splitting_subset = training_set[training_set[best_attribute] == attribute_value].drop([best_attribute], axis=1)
        sub_tree = Generate_decision_tree(splitting_subset)
        decision_tree[best_attribute][attribute_value] = sub_tree

    return decision_tree
```

그 후 반환할 decision tree를 생성해야 한다. Decision tree의 구조는 딕셔너리로 하였다. 그리디하게, top-down recursive한 방식이기에 base condition을 정해야한다. Attribute를 drop하면서 재귀가 이어질 것이므로, 만약 모든 training set의 값이 같은 class label을 갖는다면 그 값을 leaf node로 하고 재귀를 멈춘다.

또는, attribute를 모두 drop해서 더 이상 drop할게 없지만 다른 class label을 갖는다면, majority voting으로 값을 골라준다.

위 경우에 해당되지 않는다면 decision tree의 노드를 생성해주어야 한다. Information gain으로 best attribute를 고르고, 빈 딕셔너리를 선언해준다. 또한 training set에서 해당되는 attribute의 값들을 가져와서 재귀적으로 노드를 만들어 준다.

이제 이렇게 만들어진 decision tree는 다음의 구조를 갖는다.

```
{'age': {'31...40': 'yes', '<=30': {'student': {'no': 'no', 'yes': 'yes'}}, '>40': {'credit_rating': {'excellent': 'no', 'fair': 'yes'}}}}
```

위의 예시는 dt_train.txt의 decision tree이다. Age가 information gain이 제일 높았기에 처음의 분기로 채택되었고 그 이후는 재귀적으로 이어진다.

```python
def Classifier(decision_tree, test_set, class_label):
    predicted_value = []
    for test_data in range(len(test_set.index)):
        sub_tree = decision_tree
        # class_label(leaf node) 만날 때까지 탐색
        while type(sub_tree) == type(dict()):
            # 첫 번째 분류 지점
            attribute = list(sub_tree.keys())[0]

            # 분류 지점의 값들
            attribute_value = test_set.loc[test_data, attribute]

            # 탐색해야 하는 Attribute value가 분류 지점의 값들 중에 있을 경우, 1 depth만큼 전진해서 탐색
            if attribute_value in sub_tree[attribute]:
                sub_tree = sub_tree[attribute][attribute_value]

            # decision_tree에 탐색해야 하는 attribute value가 없는 경우
            else:
                nodes = list(sub_tree[attribute].values())

                candidates = []
                nextNode_candidates = []

                # leaf node가 있을 경우
                for value in nodes:
                    if type(value) == type(str()):
                        candidates.append(value)

                # leaf node가 없을 경우 대비
                for node in nodes:
                    if type(node) == type(dict()):
                        nextNode_candidates.append(node)

                # leaf node하나라도 있으면
                if len(candidates) > 0:
                    value, counts = np.unique(candidates, return_counts=True)
                    sub_tree = value[counts.argmax()]

                # leaf node가 없을 경우 그냥 다른 노드로
                else:
                    sub_tree = random.choice(nextNode_candidates)

                pass
            pass
        predicted_value.append(sub_tree)
    test_set[class_label] = predicted_value
    return test_set
```

예측한 값들은 전부 test set의 class label 자리에 인덱스로 넣어줄 수 있기 때문에 빈 리스트에 저장해준다. Test set을 한 줄씩 읽으면서 생성한 decision tree를 바탕으로 분류해준다. 기본적으로 leaf node(dict가 아닌 문자열)를 만날 때까지 탐색한다. 분류해야 하는 값이 분류 지점에 있다면 계속 재귀적으로 노드를 타고 들어간다. 그래서 문자열을 만나면 predicted_value에 추가해준다. 만약 내가 만든 모델에 부합하지 않는 경우가 되면, 두 가지로 나뉜다. 만약 leaf node가 있다면

그냥 그것으로 class label로 분류하고 그렇지 않다면 노드들 중 임의로 하나를 선택해서 leaf node를 재귀적으로 찾게 한다.

4. Testing

age	income	student	credit_rating	Class:buys_computer
<=30	low	no	fair	no
<=30	medium	yes	fair	yes
31...40	low	no	fair	yes
>40	high	no	fair	yes
>40	low	yes	excellent	no

위 사진은 dt_result.txt로 class label이 잘 매겨진 것을 확인할 수 있다.

주어진 테스트 프로그램으로 돌려본 결과는 아래와 같다.

```
PS C:\Users\mok03\OneDrive\바탕 화면\데이터 사이언스\과제\2\테스트> ./dt_test.exe dt_answer1.txt dt_result1.txt
320 / 346
```