Assignment2 Report

(Decision Tree)

2016025478 서경욱

1. 개발 환경

- **파이썬 3.8.6**

```
C:₩Users₩user>python --version
Python 3.8.6
```

2. 실행 방법

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment2 (master)
(datascience) λ python dt.py dt_train.txt dt_test.txt dt_result.txt
```

- **python dt.py (train data path) (test data path) (output file path)**

3. Summary of Algorithm

본 과제에서는 train data로 decision tree induction을 통해 classification model을 생성하고 label이 주어지지 않은 test data를 받아 tree의 policy에 따른 classification을 수행한 뒤 형식에 맞는 output file을 생성하는 프로그램을 작성하였다.

이 프로그램은 train data 경로, test data 경로, output 파일 경로를 실행 parameter로 받으며 main에서 **parameter가 알맞은 개수**로 들어왔는지 확인한다. 다음으로 train data를 dataframe 형식으로 읽어온 뒤 attribute들을 추출하고, 마지막 attribute인 class label을 저장한다. **모든 sample을 가지는** root node를 생성하고 build tree함수를 통해 tree를 recursive하게 생성한다. 이 때 test attribute의 선택은 **gain ratio**를 이용했다. Gain ratio가 가장 높은 attribute를 test attribute로 선택하고 attribute의 리스트에서 제거해준 뒤 test attribute의 case마다 child node를 생성하여 tree를 계속 build하도록 했다. Tree의 branch가 끝나는 지점은

1. 내려온 sample들이 모두 같은 class label일 때
2. sample이 없을 때

3. 남은 attribute가 없을 때

이다. 2일때는 **parent node의 data의 class label 중 가장 많은 label을 차지**하는 label로 분류했다. 3일때는 **현재 node의 data 중 가장 많은 label을 차지하는 label**로 분류했다.

Tree의 Test에서는 test data를 dataframe으로 읽어온 뒤 **class label column을 추가**하고 Nan값으로 초기화 해주었다. 그리고 dataframe의 한 행마다 classification을 진행하며 class label의 값을 넣어주었다. 만약 **Node의 classification값을 가지는 leaf**에 도달했다면 해당 값으로 class label을 넣어주고 아니라면 Node의 자식들 중 Node의 test attribute의 값에 해당하는 Node로 현재 Node를 갱신하여 recursive하게 확인하게 했다. **만약 생성된 tree의 rule에 해당하지 않아 내려갈 child node가 없다면**

1. 현재 node에 data가 있다면 그 중 가장 많은 class label의 label로 분류
2. 현재 node에 data가 없다면 전체 data중 가장 많은 class label의 label로 분류

하도록 했다.

그 후 분류한 dataframe을 output 파일 경로의 파일로 형식에 맞게 생성해주었다.

4. Detailed description of each function

```
class Node:
    def __init__(self, data):
        self.data = data
        self.parent = None
        self.test_attr = None
        self.test_info = None
        self.classification = None
        self.children = []
```

Tree를 구성하는 Node

- data: sample들을 저장
- parent: parent Node를 저장
- test_attr: 해당 node에서 child Node로 분류한 test attribute를 저장
- test_info: 해당 node에서 test_attr로 분류된 test_attr의 값을 저장

- classification: leaf의 경우 분류한 class label의 값을 저장
- children: child Node들을 저장

```
147 if __name__ == '__main__':
148     # getting parameters
149     training_path = sys.argv[1]
150     test_path = sys.argv[2]
151     output_path = sys.argv[3]
152 
153     # check correct parameter
154     if len(sys.argv) != 4:
155         print("Insufficient arguments")
156         sys.exit()
157 
158     df_train = pd.read_table(training_path)
159     attrs = df_train.columns
160     class_label = attrs[-1]
161     attrs = attrs[:-1]
162 
163     root = Node(df_train)
164     build_tree(root, df_train, class_label, attrs)
165 
166     df_test = pd.read_table(test_path)
167     df_test[class_label] = np.nan
168 
169     tree_test(df_test)
170     df_test.to_csv(output_path, sep='\t', index=False)
```

- main 함수
- 149~156 line에서 실행 parameter를 저장하고 올바른 개수의 parameter가 들어왔는지 확인한다
- 158~ 161 line에서 train data를 dataframe으로 받아오고 attribute(class label 제외)들, class label을 저장한다.
- 163,164 line에서 모든 sample을 가지는 root Node를 생성하고 **build_tree 함수를 통해 decision tree induction**을 수행한다.
- 166~170 line에서 test data를 dataframe으로 받아오고 class label column을 추가한 뒤 Nan값으로 초기화해준다. 그리고 **tree_test 함수를 통해 decision tree를 이용해 class label의 값을 넣어주고** output file로 저장해준다.

```python
def information_gain(data, class_label, attr):
    total_entropy = 0
    labels, label_cnts = np.unique(data[class_label], return_counts=True)
    for i in range(len(labels)):
        if label_cnts[i] != 0:
            total_entropy += -((label_cnts[i]/np.sum(label_cnts)) *
                               np.log2(label_cnts[i]/np.sum(label_cnts)))

    weighted = 0
    attr_samples, attr_cnts = np.unique(data[attr], return_counts=True)

    for i in range(len(attr_samples)):
        for j in range(len(labels)):
            p_ij = (len(data[((data[class_label] == labels[j]) & (
                data[attr] == attr_samples[i]))])) / attr_cnts[i]
            if p_ij != 0:
                weighted += \
                    - (attr_cnts[i]/np.sum(attr_cnts)) * p_ij * np.log2(p_ij)

    return total_entropy - weighted


def gain_ratio(data, class_label, attr):
    gain = information_gain(data, class_label, attr)

    attr_samples, attr_cnts = np.unique(data[attr], return_counts=True)
    split_info = 0
    for i in range(len(attr_samples)):
        if attr_cnts[i] != 0:
            split_info += -((attr_cnts[i]/np.sum(attr_cnts)) *
                            np.log2(attr_cnts[i]/np.sum(attr_cnts)))

    return gain / split_info
```

- information gain을 구하는 함수, gain ratio를 구하는 함수
- information gain 함수는 numpy의 unique 함수를 이용해 개수들을 count하며 수식에 맞게 entropy값을 구해주고 total entropy와 attribute에 따른 weighted average 값의 차를 구해 information gain을 구현했다.
- gain ratio 함수는 split info 값을 계산하여 information gain을 normalize해주는 것을 구현했다. (**attribute가 가질 수 있는 value 수가 더 큰 attribute에 점수를 더 주는 것을 보정**)

```python
def build_tree(cur_node, data, class_label, rest_attr):
    global root

    # sample이 모두 같은 class label
    if len(np.unique(data[class_label])) == 1:
        if cur_node.parent:
            cur_node.test_attr = cur_node.parent.test_attr
        cur_node.classification = np.unique(data[class_label])[0]
        return root
    # sample이 없을 때
    elif len(data) == 0:
        labels, counts = np.unique(
            cur_node.parent.data[class_label], return_counts=True)
        cur_node.classification = labels[counts == max(counts)][0]
        return root
    # 남은 attr이 없을 때
    elif len(rest_attr) == 0:
        labels, counts = np.unique(data[class_label], return_counts=True)
        cur_node.classification = labels[counts == max(counts)][0]
        return root
```

- build_tree 함수의 첫 부분
- top-down으로 내려가고 있는 중인 현재 노드, node로 내려온 data, class label, 남은 attribute를 parameter로 받는다.
- 함수의 시작 부분에 tree induction의 종료 조건이 있다. **sample이 모두 같은 class label의 경우** 현재 node가 leaf node가 되고, classification 값을 node가 가지는 하나의 class label 값으로 저장한다. **sample이 없을 경우** parent node의 data에서 가장 많은 label을 classification으로 설정한다. **남은 attribute가 없어** 더 이상의 branch가 불가할 경우 현재 node의 data 중 가장 많은 label을 classification으로 설정한다. 세 경우 모두 return하며 함수를 종료하며 branch를 멈춘다.

```
80      gain_ratios = [[gain_ratio(data, class_label, rest_attr[i]), rest_attr[i]]
81                     for i in range(len(rest_attr))]
82      test_attr = max(gain_ratios)[1]
83      cur_node.test_attr = test_attr
84
85      divided = np.unique(data[test_attr])
86      rest_attr = rest_attr.drop(test_attr)
87
88      for i in range(len(divided)):
89          child_Node = Node(data[data[test_attr] == divided[i]])
90          child_Node.test_info = divided[i]
91          child_Node.parent = cur_node
92          cur_node.children.append(child_Node)
93          build_tree(child_Node, child_Node.data, class_label, rest_attr)
94
95      return root
```

- build_tree 함수의 두번째 부분
- 함수 초기 조건에 걸리지 않아 **branch를 계속할 경우**의 case이다. gain ratio를 남은 attribute에 대해 모두 계산하고 저장한다. 그 중 가장 큰 attribute를 test attribute로 설정하고 현재 node의 test attribute로 설정한다.
- 85,86 line에서 test attribute로 나눌 수 있는 case들을 저장하고 test attribute를 rest_attr에서 제거한다.
- 88~93 line에서 test attribute로 나눌 수 있는 case들에 대해 data를 나누고 해당 data를 가지는 child Node를 생성한다. child Node의 나눠진 case를 test_info에 저장하고, parent node를 설정하고 현재 node의 children에 추가한다. 그리고 build_tree 함수를 통해 child Node에서의 branch를 이어나간다.

```python
def tree_test(df_test):
    global root

    for i in range(len(df_test)):
        cur_node = root
        sample = df_test.iloc[i]

        while True:
            if cur_node.classification:
                df_test[class_label][i] = cur_node.classification
                break
            else:
                test_attr = cur_node.test_attr
                done = False
                for j in range(len(cur_node.children)):
                    if sample[test_attr] == cur_node.children[j].test_info:
                        cur_node = cur_node.children[j]
                        break
                    # 분류 안되는 것
                    elif j == len(cur_node.children)-1 and sample[test_attr] != cur_node.children[j].test_info:
                        if len(cur_node.data[class_label]) > 0:
                            labels, counts = np.unique(
                                cur_node.data[class_label], return_counts=True)
                        else:
                            labels, counts = np.unique(
                                df_train[class_label], return_counts=True)
                        df_test[class_label][i] = labels[counts == max(
                            counts)][0]
                        done = True

                if done:
                    break
```

- test data의 class label을 결정해주는 tree_test 함수이다.
- 이 함수는 test data의 각 sample에 대해 반복문으로 test를 수행한다.
- 각 sample은 root Node에서 출발해서 만약 leaf node에 도달하면 해당 Node의 classification으로 class label을 설정한다.
- 현재 sample이 도달한 Node가 leaf가 아니라면 현재 Node의 자식 Node를 순회하며 해당 Node의 test attirbute의 case로 branch된 자식 Node로 현재 Node를 설정하고 다시 반복문을 수행한다.
- **만약 자식 Node를 모두 순회했으나 일치하는 자식 Node의 case가 없다면**

➔ 현재 node의 data가 있으면 data가 가장 많이 가지는 label로 sample을 분류한다.

➔ 현재 node의 data가 없다면 parent Node의 data가 가장 많이 가지는 label로 sample을 분류한다.

5. 실행 결과

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment2 (master)
(datascience) λ python dt.py dt_train.txt dt_test.txt dt_result.txt
dt.py:122: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_test[class_label][i] = cur_node.classification
C:\Users\user\Envs\datascience\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)

C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment2 (master)
(datascience) λ dt_test.exe dt_answer.txt dt_result.txt
5 / 5
```

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment2 (master)
(datascience) λ python dt.py dt_train1.txt dt_test1.txt dt_result1.txt
dt.py:122: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_test[class_label][i] = cur_node.classification
C:\Users\user\Envs\datascience\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)

C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment2 (master)
(datascience) λ dt_test.exe dt_answer1.txt dt_result1.txt
323 / 346
```