Assignment#2 Decision Tree

2017029970 우원진

알고리즘 설명

1. 먼저 Decision Tree를 사용하는데 노드에서 각 데이터에 Main_attribute를 구하기위해 GainRatio를 사용했다.

2. 거의 모든 중요한 부분은 Class 내부함수와 Class초기화 함수에 정의 되었다.

3. Tree를 만들때 사용한 방법은 DFS의 형태로 만들어 졌다.

4. 처음 Info Gain을 구할때 Pdf를 보고 구현했는데, 어쩌다 보니 2개 짜리 Info Gain으로 만들어서 정확도가 높게 나오지 않았다. 그래서 뭐가 잘못되었는지 엄청 찾아보았는데 찾지 못다가 Info Gain을 잘못 구했다는것을 알게 되었다.....

코드 설명

```
def calculate_info(p, n):  # n : 총갯수,  p : 원소 갯수
    if p == 0 or n == 0:
        return 0
    arg1 = p/n
    return round(-arg1*np.log2(arg1), 10)
```

Info를 구할때 –(시그마(p*log(p))를 사용하는데 p는 여기서 전체에 대해 얼마나 비중을 차지하는지 즉 확률이다. 여기서 처음 잘못 구했던 이유가 p를 긍정의 갯수, n을 긍정의 갯수로 두고 함수를 만들었다가 바이너리가 아닌 상황에서는 작동하지 않아 에러가 많이 발생했다..

```
class LeafNode:
    def __init__(self, result, name):
        self.result = result
        self.name = name
        # print(result, "LeafNode 추가")
```

나는 리프노드와 리프노드가 아닌 노드를 나누어 주었다.
여기서 result는 Label의 값이고, name은 부모 노드에서 어떤값에 대응 하는지 이다.
예를들어 age 가 <30, 30…40, 4, >40 일때 name은 <30일 수도, 30…40 일수도, >40 일수도 있는 것이다.

```
class Node:
    # Class Variable : data, attr_arr, main_attr, data_split_criteria, node_arr
    def split_info(self, attr):
        all_cnt = self.data.shape[0]
        temp_data = self.data.groupby(attr).size().values
        split = 0
        for value in temp_data:
            split += -value/all_cnt*np.log2(value/all_cnt)
        return split
```

이제부터 노드 Class 에 관한 내용이다. 먼저 split_info 함수는 Gain Ratio를 사용하기위해 해당 attribute에 대해 split_info 값을 구하는 함수이다.

```
    def get_all_data_gain(self):
        all_df_about_label = self.data.groupby(self.attr_arr[-1])
        all_df_about_label_cnt = all_df_about_label.size().values
        all_cnt = self.data.shape[0]

        if len(all_df_about_label_cnt) == 1:
            return 0
        else:
            info_all = 0
            for val in all_df_about_label_cnt:
                info_all += calculate_info(val, all_cnt)
            return info_all

    def calculate_gain(self, attr):
        attr_df = self.data[[attr, self.attr_arr[-1]]]
        attr_values = attr_df[attr].unique()
        attr_all_cnt = attr_df.groupby(attr).size().values.sum()
        attr_gain = 0
        for value in attr_values:  # 31...40, <=30, >40
            df_about_value = attr_df[attr_df[attr] == value]
            df_about_value_count = df_about_value.shape[0]
            df_about_value = df_about_value.groupby(
                self.attr_arr[-1]).size().values
            if len(df_about_value) == 1:
                attr_gain += 0
            else:
                for val in df_about_value:
                    attr_gain += round(df_about_value_count/attr_all_cnt *
                                       calculate_info(val, df_about_value_count), 10)
        return attr_gain
```

get_all_data_gain 함수는 현재 노드 자신이 가진 데이터에 대해 Info를 구하는 것이다.
그리고 calculate_gain(self, attr)함수는 현재 노드의 데이터에서 해당 attribute에 대해 info를 구하는 것이다. 이는 __init__함수에서 attribute마다 gain을 구해 gain이 가장큰 main_attribute를 구할때 사용되는 함수이다.

```
def __init__(self, data, attr_arr, name):
    self.data = data  # 각 노드마다 가지는 training data
    self.attr_arr = attr_arr  # 각 노드마다 가지는 남은 attribute array
    self.name = name
    if not self.data[self.attr_arr[-1]].shape[0]:
        print("What the Fuck!!")
        return
    all_gain = self.get_all_data_gain()
    max_gain = -1
    max_gain_attr = None
    for i in range(len(self.attr_arr)-1):
        attr_gain = round(
            (all_gain - self.calculate_gain(self.attr_arr[i]))/self.split_info(self.attr_arr[i]), 10)
        if(attr_gain >= max_gain):
            max_gain = attr_gain
            max_gain_attr = self.attr_arr[i]
    self.main_attr = max_gain_attr
    self.data_split_criteria = self.data[self.main_attr].unique()
    # print("일반 Node 추가")
    # print("Node name : ", self.name)
    # print("Main Attr : ", self.main_attr)
    # print(self.data)
    next_node_attr_arr = copy.deepcopy(self.attr_arr)
    next_node_attr_arr = np.delete(next_node_attr_arr, np.where(
        next_node_attr_arr == self.main_attr))
```

이제 가장 중요한 __init__ 함수이다. 이 함수는 데이터와, attribute 리스트, 이름을 Argument로 받는다.

만약 데이터가 하나도 없으면 바로 리턴한다. 그리고 위에서 설명한 함수들을 사용해 각 Attribute 마다 Gain Ratio를 구하고 가장 큰 Gain Ratio에 해당하는 Attribute를 그 노드의 main_attribute로 설정한다. 그리고 그 Attribute에 해당하는 value들의 unique한 리스트를 만들어 data_split_criteria에 담는다. 그리고 array와 같은 변수는 주소값이 할당 되기 때문에 deepcopy를 통해 next_node_attr_arr이라는 변수에 담고, 현재 사용한 attribute 즉 main_attribute를 next_node_attr_arr 에서 삭제한다. next_node_attr_arr 변수는 자식 노드들의 attr_arr에 담기게 될것이다.

```
    self.node_arr = []
    for criteria in self.data_split_criteria:
        new_data = self.data.groupby(self.main_attr).get_group(criteria)
        next_node_data = new_data.drop(columns=[self.main_attr])
        # 모든 라벨이 같을때 LeafNode
        if(len(next_node_data[next_node_attr_arr[-1]].unique()) == 1):
            result = new_data[next_node_attr_arr[-1]].values[0]
            temp_node = LeafNode(result, criteria)
            self.node_arr.append(temp_node)
        elif(len(next_node_attr_arr) == 1):  # 더이상 attribute가 없을때 LeafNode
            result_dict = {}
            for res in next_node_data[next_node_attr_arr[-1]].values:
                if res in result_dict:
                    result_dict[res] += 1
                else:
                    result_dict[res] = 1
            result = None
            max_cnt = 0
            for key in result_dict.keys():
                if result_dict[key] >= max_cnt:
                    max_cnt = result_dict[key]
                    result = key
            temp_node = LeafNode(result, criteria)
            self.node_arr.append(temp_node)
        elif(not new_data.shape):  # 데이터가 하나도 없을때 -> LeafNode를 어떻게 할지 고민
            temp_data = self.data[self.attr_arr[-1]].size()
            print(temp_data)
        else:  # 일반 node추가
            temp_node = Node(next_node_data, copy.deepcopy(
                next_node_attr_arr), criteria)
            self.node_arr.append(temp_node)
```

self.node_arr 은 main_attribute에 의해 나누어지게 되어 생기는 child Node들을 담는 리스트 이다.

여기서 new_data 변수에는 현재 노드가 가진 데이터에서 main_attribute에 의해 그룹하 되어 attribute 의 value 별로 데이터를 가져와서 담은 변수 이다. 그리고 next_node_data 변수는 new_data의 data에서 main_attribute에 해당하는 column을 지운 것이다. 이는 child node의 data로 넘겨진다. 여기서 Child Node들이 생성되는데 자식 노드 들은 Leaf 노드 일수도있고 아닐수도 있다. 그렇기 때문에 조건문을 통해서 이를 나누었다. 만약 자식 노드로 보낼 데이터에서 모든 라벨이

같거나 Attribute를 모두 사용했거나, 더이상 데이터가 없으면 리프 노드로 생각했다. 이를 제외한 상황은 리프노드가 아닌 노드이다. 리프노드일때의 라벨은 만약 모든 라벨이 같으면 그 라벨로 설정하고, Attribute가 없는 상황이면 그 데이터중에서 다수결의 원칙을 적용해 라벨을 설정했다. 그리고 데이터가 없는 상황은 에러 상황으로 생각했다.

```
train_file = sys.argv[1]
test_file = sys.argv[2]
result_file = sys.argv[3]

train_df = pd.read_csv(train_file, sep="\t", dtype=str)
test_df = pd.read_csv(test_file, sep="\t", dtype=str)
# ans_df = pd.read_csv(result_file, sep="\t", dtype=str)  # for test

all_attr = train_df[train_df.columns.values[-1]].unique()
majority_all = train_df.groupby(train_df.columns.values[-1])
max_count = 0
majority_voting_result = ""
for a in all_attr:
    val = majority_all.size()[a]
    if val >= max_count:
        max_count = val
        majority_voting_result = a
```

이 부분은 pandas 라이브러리를 통해 파일을 읽고 일단 나중에 사용할 수도 있는 전체 데이터에 대한 다수결의 원칙에 의해 결정된 결과를 저장해 놓는 코드이다.

```
def traverse_node(root, data_dict):
    node = root
    while True:
        if node.__class__ == LeafNode:
            return node.result
        is_in = False
        for temp_node in node.node_arr:
            if temp_node.name == data_dict[node.main_attr]:
                node = temp_node
                is_in = True
                break
        if not is_in:  # 트레이닝 데이터셋으로 만들었을때 해당 노드가 없는경우 -> 부모노드에서 다수결로 한다 -> 실습에 물어보고 결정
            data = node.data
            attr_arr_temp = data[data.columns.values[-1]].unique()
            data = data.groupby(node.attr_arr[-1])
            max_count_temp = 0
            majority_voting_result_temp = ""
            major_arr = []
            for a in attr_arr_temp:
                val = data.size()[a]
                if val > max_count_temp:
                    max_count_temp = val
                    majority_voting_result_temp = a
            # return "e" # for error data
            return majority_voting_result_temp
```

이 함수는 test하거나, 어떤 데이터가 들어왔을때 만들어진 트리를 타고 들어가 어떤 라벨에 해당하는지 결정하는 함수이다. 만약 트레이닝 데이터셋에 없는 데이터가 들어왔을때는(에러 데이터) 에러가 발생한 부분의 노드에서 다수결의 원칙을 적용해 라벨을 반환해줬다.

```python
def find_result(root, test_data, result_attr):
    attr_arr = test_data.columns.values
    test_data[result_attr] = ""
    for i in range(test_data.shape[0]):
        data = test_data.iloc[i]
        data_dict = {}
        for attr in attr_arr:
            data_dict[attr] = data[attr]
        result = traverse_node(root, data_dict)
        if result == "e":
            error_data.append(data)
        test_data.at[i, result_attr] = result
    test_data.to_csv(result_file, sep="\t", index=False)
    # 테스팅 코드
    # my_ans = test_data[result_attr]
    # real_ans = ans_df[result_attr]
    # all_cnt = 0
    # correct_cnt = 0
    # for i in range(test_data.shape[0]):
    #     all_cnt += 1
    #     if(my_ans.iloc[i] == real_ans.iloc[i]):
    #         correct_cnt += 1
    # print(f"{correct_cnt}/{all_cnt} -> {round(correct_cnt/all_cnt*100, 3)}% accuracy")
```

이 함수는 이제 위에서 설명한 함수를 가지고 새로운 데이터 프레임을 만들어 pandas를 통해 결과 파일을 만들어내는 함수이다.

실행과정

```
uwonjin-ui-MacBookPro:Assignment#2 woowonjin$ python3 dt.py dt_train1.txt dt_test1.txt dt_result1.txt
```

결과

먼저 에러에서도 다수결의 원칙에 의해 값을 내줬을때의 결과이다.

```
uwonjin-ui-MacBookPro:Assignment#2 woowonjin$ python3 dt.py dt_train.txt dt_test.txt dt_answer.txt
5/5 -> 100.0% accuracy
uwonjin-ui-MacBookPro:Assignment#2 woowonjin$ python3 dt.py dt_train1.txt dt_test1.txt dt_answer1.txt
318/346 -> 91.908% accuracy
```

에러값은 버렸을때의 결과.

```
uwonjin-ui-MacBookPro:Assignment#2 woowonjin$ python3 dt.py dt_train.txt dt_test.txt dt_answer.txt
5/5 -> 100.0% accuracy
uwonjin-ui-MacBookPro:Assignment#2 woowonjin$ python3 dt.py dt_train1.txt dt_test1.txt dt_answer1.txt
302/319 -> 94.671% accuracy
```