Assignment-2 (Decision Tree)

2016025105
컴퓨터공학과
강재훈

(i) Summary of your algorithm

Decision Tree는 전형적인 분류 모델입니다. Supervised learning을 통해 모델을 construction 합니다. Decision tree를 induction 하는 알고리즘의 특징에는 greedy하다, top-down 방식으로 진행된다, divide and conquer 문제이다, recursive하게 진행이 된다는 특징이 있습니다. 이러한 특징들을 반영해 해당 알고리즘을 설명할 수 있는데 현 시점의 node에서 가지고 있는 data를 가장 homogeneous하게 나누는 test attribute를 선정하고 (greedy) data를 나눕니다. 이후 나눠진 각각의 data에 대해서도 위와 같은 방식을 적용해 점차 아래로 (top-down, divide and conquer) 내려가며 tree를 완성시킵니다. Recursion을 멈추는 시점은 총 3가지로 첫번째는 더이상 나누는 기준이 될 수 있는 attribute가 없을 때. 두번째로는 data 내의 모든 object의 class label이 같을 때. 마지막으로는 더이상 data 가 남아있지 않을때 입니다. 코드에서 세경우 모두 구현하였습니다.
test attribute를 선정하는 방식으론 information gain, gain ratio, gini index 등이 있는데 여러 알고리즘들 모두 data를 최대한 homogenuous 하게 나누는 것에 관심이 있다는 공통점이 있습니다. 제 구현된 제 코드는 gain ratio 방식을 채택하였습니다. Gain ratio 방식은 기본적인 information gain에 크기에 대한 normalization을 해주는 알고리즘으로 생각할 수 있습니다. 하지만 gain ratio 역시 split 된 set 들의 크기가 동일하지 않고 한쪽이 크고 한쪽이 작은 쪽으로 나눠지도록 test attribute를 선정하는 경향이 있다는 단점이 있습니다.
Decision Tree의 장점으로는 deep learning과는 다르게 설명이 가능 하다는 점. Learning speed가 굉장히 빠르다는 점. 이해하기 쉬운 classification rule로 바꿀 수 있다는 점. SQL query를 이용할수 있다는 점. Classification accuracy가 다른 모델들과 comparable 하다는 점이 있습니다.

.

(ii) Summary of your algorithm

- class InternalNode, LeafNode

Decision Tree의 internal node와 leaf node를 구현하기 위해 작성한 class 입니다. Internal node는 attribute를 담고 있는 node이고, leaf node는 class label을 담고 있는node 입니다. <__str__ 메소드는 디버깅을 위해 작성되었습니다.>

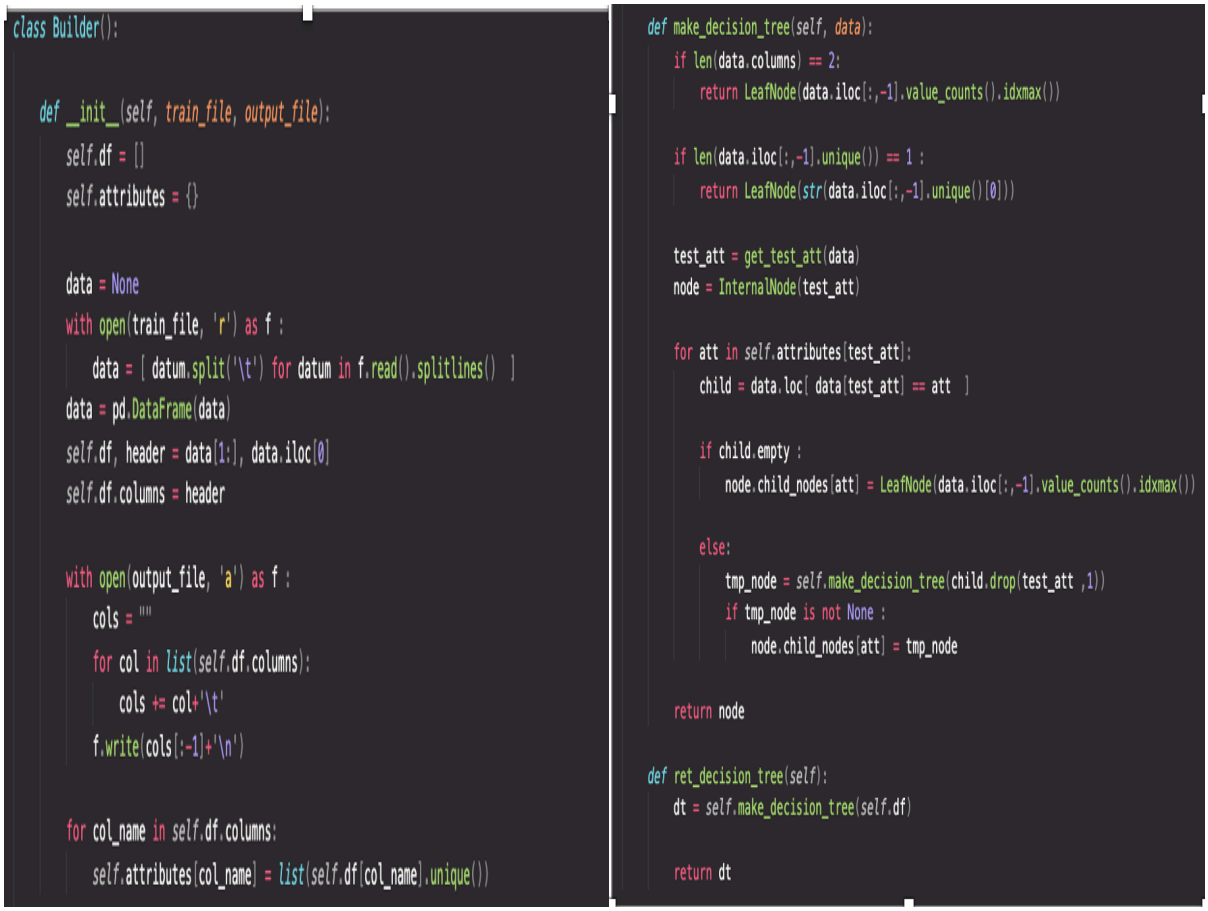
```
class InternalNode():
    def __init__(self, attribute):
        self.attribute = attribute
        self.child_nodes = {}

    def __str__(self):
        ret = str(self.__dict__)
        return ret


class LeafNode():
    def __init__(self, label) :
        self.label = label

    def __str__(self):
        ret = str(self.label)
        return ret
```

- class Builder
Decision Tree를 구축해주는 class 입니다.

__init__ 함수 == 객체를 만듦과 동시에 train data를 읽어와 df에 저장하고 attributes에 각각의 attribute와 value들을 dictinary 형태로 저장합니다. 이후 attribute와 label의 이름을 output file에 작성해줍니다.

make_decision_tree 함수 == data가 주어지면 해당 data를 가지고 decision tree를 build 해주는 함수입니다. 앞서 기술한 tree의 특징들을 적용하여 만들었습니다. induction 하는 알고리즘은 recursive하다는 특징을 살려 recursive 하게 tree를 build 하게 만들었습니다. 또한 recursive가 종료되는 3가지 조건을 모두 반영하였으며 Internal Node가 아닌 Leaf node가 반환되게 만들었습니다. 또한 후술할 get_test_att 함수를 사용해 test attribute를 gain ratio로 선정하였으며 만약 leaf에 도달한 data가 homogeneous 하지 않다면 majority voting 방식을 취해 label을 결정하도록 하였습니다.

ret_decision_tree 함수 == Builder는 decision tree만 만들고 실제 작동을 분리하기 위해 make_decision_tree 함수를 통해 만든 decision tree를 반환해주는 함수입니다.



```
class Builder():

    def __init__(self, train_file, output_file):
        self.df = []
        self.attributes = {}

        data = None
        with open(train_file, 'r') as f :
            data = [ datum.split('\t') for datum in f.read().splitlines()  ]
        data = pd.DataFrame(data)
        self.df, header = data[1:], data.iloc[0]
        self.df.columns = header

        with open(output_file, 'a') as f :
            cols = ""
            for col in list(self.df.columns):
                cols += col+'\t'
            f.write(cols[:-1]+'\n')

        for col_name in self.df.columns:
            self.attributes[col_name] = list(self.df[col_name].unique())
```

```
    def make_decision_tree(self, data):
        if len(data.columns) == 2:
            return LeafNode(data.iloc[:,-1].value_counts().idxmax())

        if len(data.iloc[:,-1].unique()) == 1 :
            return LeafNode(str(data.iloc[:,-1].unique()[0]))

        test_att = get_test_att(data)
        node = InternalNode(test_att)

        for att in self.attributes[test_att]:
            child = data.loc[ data[test_att] == att  ]

            if child.empty :
                node.child_nodes[att] = LeafNode(data.iloc[:,-1].value_counts().idxmax())

            else:
                tmp_node = self.make_decision_tree(child.drop(test_att ,1))
                if tmp_node is not None :
                    node.child_nodes[att] = tmp_node

        return node

    def ret_decision_tree(self):
        dt = self.make_decision_tree(self.df)

        return dt
```

- def info, gain_ratio, get_test_att

info 함수 == expected information 을 계산해주는 함수 입니다.

gain_ratio 함수 == info 함수로 구해진 expected information을 이용해 gain ratio 구합니다.

get_test_att 함수 == gain ratio를 비교해 test attribute를 선정하는 함수입니다.

< 계산식들과 원리는 5주차 이론 강의를 참고하였습니다. >

```
def info(val):
    val_sum, info = val.sum(), 0

    for num in val :
        if num :
            info -= (num/val_sum) * math.log(num/val_sum)
        else :
            return 0
    return info


def gain_ratio(table):
    all_val_sum = table.values.sum()
    wei_avg, spl_info = 0, 0

    for val in table.values :
        val_div_sum = val.sum()/ all_val_sum
        wei_avg += val_div_sum * info(val)
        spl_info -= val_div_sum * math.log(val_div_sum) / math.log(2)

    ret = wei_avg / spl_info

    return ret


def get_test_att(data):
    cand_atts = dict()

    att_names = data.columns[:-1]
    label = data[data.columns[-1]]

    for att_name in att_names :
        cross_table = pd.crosstab(data[att_name], label)
        cand_atts[att_name] = gain_ratio(cross_table)

    ret = min(cand_atts.keys(), key = lambda x : cand_atts[x])

    return ret
```

- def classification, classification_using_dt

classification 함수 == decision tree를 따라 내려가며 label을 찾습니다.

Classification_using_dt 함수 == decision tree를 이용해 classification합니다. 찾은 label을 기존 정보들과 합쳐 result 파일에 기록합니다.

< 계산식들과 원리는 5주차 이론 강의를 참고하였습니다. >

```
def classification(node, row):
    if isinstance(node, LeafNode):
        return node.label
    else :
        return classification( node.child_nodes[row[node.attribute]], row   )

def classification_using_dt(dt, test_file, output_file):
    test_df = None

    data = None
    with open(test_file, 'r') as f :
        data = [ datum.split('\t') for datum in f.read().splitlines()  ]
    data = pd.DataFrame(data)
    test_df, header = data[1:], data.iloc[0]
    test_df.columns = header

    with open(output_file, 'a') as f :
        for i, row in test_df.iterrows():
            result = list(row.values)
            result.append( classification(dt, row) )

            sentence = ""
            for word in result :
                sentence += word+'\t'
            f.write(sentence[:-1]+'\n')
```

- def bfs

Bfs 함수 == tree를 탐색하며 decision tree를 출력합니다. < debug 용 >

```
def bfs(root):
    q = deque()
    q.append(root)

    while q :
        for i in range(len(q)) :
            t = q.popleft()
            if isinstance(t, LeafNode):
                print(t.label,"|"+'\t', end= " ")
            else :
                print(t.attribute, t.child_nodes.keys(), "|"+'\t' , end = " ")
                for child in t.child_nodes :
                    q.append(t.child_nodes[child])

        print()
```

(iii) Instructions for compiling your source codes at TA’s computer

python version == Python 3.9.4
실행법 == python dt.py train파일명 test파일명 결과파일명
실행법2 == dt_test.exe 정답파일명 결과파일명

(iii) Any other specification of your implementation and testing