Programming Assignment #2

2018009870 이찬호

1. Instructions for compiling your source codes at TA's computer

window : dt.py (train.txt) (test.txt) (output.txt) 형식으로 작성
linux : python dt.py (train.txt) (test.txt) (output.txt) 형식으로 작성

2. Summary of your algorithm

tree의 노드를 구성할 때, child 노드를 저장하기 위하여 list를 활용하였다. 또한 어떠한 feature를 사용하여 child로 나누었는지를 저장하기 위해서 category라는 attribute를 활용하였다.

entropy 함수는 input으로 pandas의 DataFrame 형식을 받고, 어떤 feature를 기준으로 계산을 할지 받는다.

Information gain, gain ratio의 경우 input으로 pandas의 DataFrame 형식을 받고, 어떤 feature를 기준으로 계산을 할지에 대한 정보와 class label을 받는다.

gini의 경우 input으로 pandas의 DataFrame 형식을 받고, 어떤 feature를 기준으로 계산할지에 대한 정보를 받는다.

이 때 gini의 경우, 특정 feature의 종류가 여러 개일 경우, 이를 2개로 나누어야 한다. 따라서 마지막의 feature와 나머지로 나누어서 gini index를 계산하였다. 하지만 이는 최적화를 하지 않은 것이기 때문에 모든 가지 수를 계산하여 가장 좋은 것을 고르는 최적화가 필요하다.

tree라는 함수는 decision tree를 생성하는 함수이다. node class 중 모든 data를 가지고 있는 것을 input으로 받고 data와 category의 목록을 받습니다. 만약 데이터가 없거나, 데이터의 class label이 1개일 경우, 완벽하게 분류된 것이므로 return을 한다. 그렇지 않은 경우, index에 따라서 가장 잘 나누는 feature를 찾은 후, 이를 가지고 나눈 다음, 노드의 child에 각각 나눈 데이터를 소유한 노드를 만들어서 집어넣고, 다시 tree라는 함수를 각각의 child 노드에 대해서 다시 호출하여 재귀적으로 tree를 만들어간다.

fit이라는 함수는 decision tree를 가지고 새로 들어온 데이터들의 class label을 결정하는 함수이다. 처음으로 가장 상위의 노드와 category의 목록, 새로 들어온 데이터를 입력으로 받는다. 만약 특정 노드에서 child 노드로 가기 위해서는 feature가 일치해야하는데, 그렇지 않은 경우 그 노드의 데이터에서 majority를 계산하여 class label을 결정하였다.

3. Detailed description of your codes

```
# tree node class
class Node:
    def __init__(self,value):
        self.child = list() # child가 몇 개인지 확신할 수 없으므로 이를 담을 list로 생성
        self.data = value # 특정 노드에서 child 노드로 나누어지기 전 data
        self.category = None # 어떤 feature를 가지고 child 노드를 나누었는지 저장
        self.flag = None # feature 중 어떤 특징을 가지고 child 노드가 되었는지 저장
# entropy는 class label과  feature에서 계산을 할 수 있어야 하므로 data와 feature를 받는다.
def entropy(data,target):
    answer = 0
    data = data.loc[:,target] # 특정 feature의 column을 뽑아낸다.
    elements, count = np.unique(data,return_counts=True) # column에 어떤 종류의 feqture가 존재하고, 얼마나 존재하는지 알기 위해서 numpy의 unique를 활용
    for i in range(len(elements)):
        answer += count[i]/np.sum(count)*np.log2(count[i]/np.sum(count)+np.finfo(float).eps) # 계산한 feature의 종류와 개수를 가지고 entropy 계산. 이 때, log에 0이 들어가지 않도록 하기 위해서 epsilon을 더해서 이를 방지하였음.
    return -answer

def InformationGain(data,target1,target2):
    # target1 is label
    # target2 is attribute

    answer = 0
    # 나누기 이전 data의 entropy를 계산
    total_entropy = entropy(data,target1)
    # 나눌 feature의 column을 슬라이싱
    column = data[target2]
    # column에 어떤 종류의 feqture가 존재하고, 얼마나 존재하는지 알기 위해서 numpy의 unique를 활용
    weight,count = np.unique(column,return_counts=True)
    # 각각의 데이터를 나눈 다음 그에 대한 entropy를 계산한 다음 가중치 평균을 구해 기존의 total entropy에서 빼줌.
    for i in range(len(weight)):
        # 특정 feature의 종류에 따라서 group을 나눔
        group = data.where(data[target2] == weight[i]).dropna()
        answer += count[i]/np.sum(count)*entropy(group, target1)
```

```
    answer -= total_entropy
    return -answer

def GainRatio(data,target1,target2):
    # target1 is label
    # target2 is attribute
    answer = 0
    # information gain을 계산.
    gain = InformationGain(data, target1, target2)

    column = data[target2]
    # normalization을 진행하기 위해서 feature에 포함된 종류와 개수를 측정
    split,count = np.unique(column,return_counts=True)

    for i in range(len(split)):
        answer                                                                    +=
count[i]/np.sum(count)*np.log2(count[i]/np.sum(count)+np.finfo(float).eps)
    # normalization을 진행
    return gain/(-answer)

def gini(data,target):
    answer = 0
    # 진행하려는 feature를 슬라이싱
    column = data[target]
    elements, count = np.unique(column,return_counts=True)
    gini = 1
    # 만약 feature에 포함된 종류가 2가지라면 gini index 바로 계산
    if len(elements) == 2:
        for i in range(len(elements)):
            gini -= np.power(count[i]/np.sum(count),2)
        answer += count[i]/np.sum(count)*gini
    # 1가지인 경우 gini index를 0으로 출력
    elif len(elements) == 1:
        return 0
    # 3가지 이상일 경우 gini index를 구하기 위해서 가장 뒤의 feature과 나머지 총 2가지
로 나누어서 gini index 계산
    else :
        before_count = 0
        for i in range(len(elements)-1):
            before_count += count[i]
        gini -= np.power(before_count/np.sum(count),2)
```

```
            gini -= np.power(count[-1]/np.sum(count),2)
            answer += before_count/np.sum(count)*gini
            answer += count[-1]/np.sum(count)*gini
        return answer

def tree(data,category,parent):
    # class label의 종류와 개수 파악
    column = data[category[-1]]
    elements,count = np.unique(column,return_counts=True)
    # 종료 조건
    # class label의 개수가 1개일 때는 잘 분리된 것이므로 종료
    if len(elements) == 1:
        return
    # 들어오는 데이터가 없으면 종료
    elif len(data) == 0:
        return
    # class label의 개수가 2개 이상인 경우, child 노드를 만들기 위해서 가장 적절한 feature를 찾고 tree 성장
    else :
        # find feature
        max_measure = 0
        divide = None
        # Gain Ratio를 통해서 가장 좋은 feature를 구하는 것
        # 모든 feature에 Gain ratio를 구해서 가장 높은 것을 feature로 선택
        for i in range(len(category)-1):
            measure = GainRatio(data, category[-1],category[i])
            if measure > max_measure:
                divide = category[i]
                max_measure = measure
        # grow tree
        # 가장 높은 feature를 가지고 data를 나누기
        column = data[divide]
        elements,count = np.unique(column,return_counts=True)
        # feature가 나타내는 개수에 따라서 child 노드의 개수가 정해지고 그에 따라서 새로운 tree 함수가 호출됨
        for i in range(len(elements)):
            group = data.where(data[divide] == elements[i]).dropna()
            child = Node(group)
            parent.child.append(child)
            child.flag = elements[i]
            tree(group,category,child)
```

```
        parent.category = divide

def fit(data,category,model):
    count = 0
    # node의 child node가 없을 때 까지 leaf로 이동, child node가 없는 경우, 현재의 node를 model이라는 변수에 저장
    while(model.child != list()):
        if count != 0:
            break
        # child 노드를 나눈 기준을 읽기
        divide = model.category
        # child 노드를 확인하며, 들어온 데이터의 feature와 일치하는 것이 있는지 확인
        for i in model.child:
            count += 1
            # 있다면 child를 다시 parent로 바꾸어서 child node를 확인
            # 없다면 현재의 node를 model에 저장하고 while문 종료
            if i.flag == data[divide]:
                model = i
                count = 0
                break
    # 현재 노드의 데이터들의 class label을 읽음
    column = model.data[category[-1]]
    elements, count = np.unique(column,return_counts=True)
    maximum = 0
    label = elements[0]
    # 가장 많은 class label을 찾아서 그것을 출력함.
    for i in range(len(elements)):
        if maximum < count[i]:
            maximum = count[i]
            label = elements[i]
    return label

path = os.getcwd()
# 명령어 입력을 받는 곳
command = sys.argv

try :
    if len(command) != 4:
        raise Exception("명령어가 잘못 입력되었습니다.")
except Exception as e:
```

```
    print("명령어를 문법에 맞게 사용하여 주세요. ex. dt.py dt_train.txt dt_test.txt
dt_result.txt")
    exit()

# set file stream

train = command[1]
test = command[2]
save = command[3]

TrainData = open(os.path.join(path,train),'r')
TestData = open(os.path.join(path,test),'r')
SaveResult = open(os.path.join(path,save),'w')

train = pd.read_csv(TrainData,sep="\t")
test = pd.read_csv(TestData,sep="\t")

root = Node(train)
# 입력 데이터의 feature 이름 확인
category = train.columns

# tree 성장 가장 상위 노드는 root
tree(train,category,root)

(row,column) = train.shape
# train가 잘 되었는지 확인하기 위한 코드
# train의 data를 가지고 decision tree를 통해서 결과 확인,
output = np.empty((row,1),dtype=object)
count = 0
for i in range(row):
    if train.iloc[i,-1] == fit(train.iloc[i,:],category,root):
        count += 1
print(count,'/',row)

# test data set을 가지고 decision tree를 활용해 출력 결정
(row,column) = test.shape
# 출력 데이터를 저장할 변수
output = np.empty((row,1),dtype=object)

for i in range(row):
    output[i] = fit(test.iloc[i,:],category,root)
```

```
output = pd.DataFrame(data=output)
test[category[-1]] = output

# 형식에 맞추어 데이터 저장
for i in range(len(test.columns)):
    SaveResult.write(test.columns[i])
    if len(test.columns) - 1 == i :
        SaveResult.write("\n")
    else :
        SaveResult.write('\t')

for i in range(row):
    for j in range(column+1):
        SaveResult.write(test.iloc[i,j])
        if j == column:
            SaveResult.write("\n")
        else :
            SaveResult.write("\t")

TrainData.close()
TestData.close()
SaveResult.close()
```