Decision Tree

과목명.	데이터 사이언스
담 당.	김 상 욱 교수님
제출일.	2021년 04월 13일
공과대학	소프트웨어전공
4 학년,	학 번. 2016024884
이 름.	송준익

HANYANG UNIVERSITY

목 차

1. Algorithm Description

이번 과제에서 제가 구현한 알고리즘을 간단히 정리하면 아래와 같습니다.

1. 파일을 스캔하여 배열 형태로 저장
    A. 첫번째 줄은 attribute 행렬에 따로 저장합니다.
    B. 두번째 이후의 줄은 배열 형태로 변환하여, 배열에 저장합니다.
    C. 즉, 데이터는 2차원 배열 안에 정리되게 됩니다.
2. Decision Tree 제작
    A. 먼저, 모든 데이터를 담은 root노드를 생성합니다.
    B. 노드에는 분류될 데이터가 있고, 분류된 후의 gain ratio들을 비교하여 정보 획득이 가장 큰 데이터를 기준으로 분류하게 됩니다.
    C. 이때, 분류되는 기준 데이터를 node의 label에 저장합니다.
    D. 그 후, 분류된 데이터들을 담은 노드들이 node의 children 배열에 들어가게 됩니다.
    E. 그리고, node의 각 child에 대해 이 작업을 반복합니다.
    F. max depth에 도달하거나 노드 내의 결과값의 purity가 일정 값 이상이 되면, 가장 많은 결과값을 노드의 label에 저장합니다. 이 노드들은 말단 노드가 됩니다.
3. Test case에 대하여 Decision Tree 탐색
    A. test case들을 스캔해 배열을 생성합니다.
    B. root에서 시작하여 말단 노드에 도착할 때까지 각 노드에 붙은 label을 보고 트리를 탐색합니다.
    C. 말단 노드에 도착하면, 말단 노드의 label을 출력합니다. 이것이 곧 결과값이 됩니다.

2. Code Description

이번 과제에서 Decision Tree의 구현을 위해 사용한 방법은 아래와 같습니다.

먼저, main 함수에서는 기본적인 파일 입력과 함수 호출을 수행합니다.

```
for l in lines:
    temp = l.split('₩n')
    spliter = temp[0].split('₩t')
    if(isFirst):
        attr_num = len(spliter)
        attr = spliter
        isFirst = False
        continue
    train_list.append(spliter)
    candi+=1
f.close

vars = []

for a in range(attr_num):
    vars.append([])
for i in range(candi-1):
    for j in range(attr_num):
        if(train_list[i][j] not in vars[j]):
            vars[j].append(train_list[i][j])

final = vars[attr_num-1]
root = node(train_list)
print("Start making decision tree...")

makenode(root)
```

1. file의 input data를 저장하기 위해 먼저 파일을 읽어와 첫번째 라인은 attr에, 그 이후 라인들은 train_list에 저장합니다.
2. 이때, variables의 경우를 모두 저장하기 위해 vars 배열을 사용했습니다.
3. 그 후, root를 만들어준 뒤 root에 대해 makenode함수를 실행합니다. 이 함수를 통해 decision tree를 만들게 됩니다.

```
for i in range(attr_num):
    size = len(vars[i])
    sinfo = np.zeros(size)
    divided_list = []
    for tmp in range(size):
        divided_list.append([])
    for j in range(lenlist):
        for k in range(len(vars[i])):
            if(list[j][i] == vars[i][k]):
                divided_list[k].append(j)
                sinfo[k] += 1
    for s in sinfo:
        s = float(s)/sum(sinfo)
    splitinfo = entropy(sinfo)
    ents = []
    all = 0
    if(divnum(divided_list) <= 1):
        continue
    for d in divided_list:
        all+=len(d)
    for dv in divided_list:
        for case in dv:
            nums = np.zeros(final_len)
            for f in range(final_len):
                if(list[case][attr_num] == final[f]):
                    nums[f] += 1
            ents.append(entropy(nums))
        for e in range(len(ents)):
            ents[e] /= float(all)
            ents[e] *= len(dv)
    entro = sum(ents)
    gain = origin_ent - entro
    gainratio = gain/splitinfo
    if(max_gain < gainratio):
        idx = i
        max_gain = gainratio
        answer = divided_list
```

4. Makenode 함수 내부에서 decision tree를 만드는 작업입니다. 먼저 각 attribute를 기준으로 분할한 리스트들을 저장하기 위해 divided_list를 선언합니다. 이제 각 attribute들을 기준으로 분할한 데이터들을 각 divided_list에 삽입합니다.

5. Divided_list별로 gain ratio를 계산합니다. 즉, gain을 구해준 다음 splitinfo로 나누고, 이렇게 얻은 값들 중 가장 큰 값을 찾습니다.

6. 가장 큰 gain ratio 값을 갖게 되는 divided_list를 answer에 저장합니다.

```
for child in answer:
    c_list = []
    for index in child:
        c_list.append(list[index])
    if(len(c_list) > 0):
        nod.children.append(node(c_list))
    else:
        tmp_nod = node([])
        tmp_nod.isend = True
        tmp_nod.label = findmost(final_list)
        nod.children.append(tmp_nod)
if(len(nod.children) > 0):
    for child in nod.children:
        child.depth = nod.depth + 1
        if(child.isend == False):
            makenode(child)
return
```

7. Answer에 저장된 값을 data로 갖는 node들을 생성해, 해당 node의 children으로 지정합니다.

8. 단, child의 data배열 길이가 0일 경우에는 node중 가장 많은 결과값과 동일한 결과를 지니는 단말 노드로 취급해 연산을 끝냈습니다.

9. 깊이를 1 증가시킨 뒤, 단말 노드가 아닌 child에 대해서도 makenode함수를 실행합니다.

```
def predict(test_list, result):
    f=open(result, 'w')
    for at in attr:
        f.write(at + "₩t")
    f.write("₩n")
    for testcase in test_list:
        nod = root
        for at in testcase:
            f.write(at + "₩t")
        while(nod.isend == False):
            idx = attr.index(nod.label)
            v = vars[idx].index(testcase[idx])
            nod = nod.children[v]
        f.write(nod.label)
        f.write("₩n")
    f.close
    return
```

10. Predict 함수에서는 label을 따라가며 단말 노드를 찾아 단말 노드의 label을 출력하도록 했습니다.

```python
import numpy as np
import sys
import math
from collections import Counter

bound_purity = 0.8
max_depth = 6
```

11. 해당 과제를 수행하는데 사용된 open library입니다. 또한, 해당 노드가 단말인지를 판단하는데 사용된 bound_purity와 max_depth값입니다. 해당 값에 변화를 주니 결과도 바뀌었습니다.

```python
class node:
    def __init__(self, data):
        self.data = data
        self.label = ''
        self.children = []
        self.isend = False
        self.depth = -1
        self.final_val = []
        for d in data:
            self.final_val.append(d[len(attr)-1])
        self.final_per = np.zeros(len(final))

        for i in self.final_val:
            for f in range(len(final)):
                if(i == final[f]):
                    self.final_per[f] += 1
        self.final_sum = sum(self.final_per)
        for f in range(len(final)):
            if(self.final_per[f] >= float(self.final_sum)*bound_purity and self.final_sum > 0):
                self.isend = True
                self.label = final[f]
```

12. Decision tree 구현에 사용된 node 클래스입니다. 선언된 직후 결과값의 순도가 bound purity 이상이면 즉시 단말 노드로 분류됩니다.

3. Compiling Description

```
PS C:\Users\User\Source\repos\DataScience\Project2_decisiontree\dt\dt> py dt.py dt_train.txt dt_test.txt dt_result.txt
Start making decision tree...

Decision tree made succesfully. Start Testing...

Test complete! Output file dt_result.txt created.
```

Python을 사용했으므로, 위와 같이 입력하면 아주 잘 돌아갑니다.

다만, 실행을 확인하기 위해 문장을 출력하도록 했습니다.

4. Specifications and Testing

해당 과제는 bound_purity와 max_depth에 어떤 값을 넣어주느냐에 따라서 결과가 달라지는 현상을 보였습니다. 과제에서 주어진 데이터를 이용해 비교한 결과값은 아래와 같습니다.

Bound Purity	Test case accuracy
0.9	277/346
0.8	279/346
0.7	235/346

Max_depth를 고려하지 않고 bound purity를 조절했을 때, 각각 0.9, 0.8, 0.7일때의 값입니다. Bound purity를 0.8로 설정했을 때 최적의 결과를 보이는 것을 확인할 수 있습니다.

Max depth	Training set accuracy	Test case accuracy
3	1131/1382	271/346
4	1156/1382	272/346
5	1260/1382	279/346
6	1380/1382	279/346

Bound purity = 0.8일 때 max depth에 따른 정확도의 변화입니다. Max depth가 6일 때 Training set과 Test set 둘 다에서 높은 정확도를 보입니다. 6보다 큰 값을 넣었을 때는 더 이상 값이 변하지 않았습니다.

즉, bound purity = 0.8, max depth = 6일 때 최적의 결과값을 얻을 수 있었습니다. 따라서 제가 제출한 코드에는 값이 그렇게 고정되어 있습니다.