기계학습원론 HW6

2021312738 소프트웨어학과 김서환

1. You have a set {T,T,T,T,T,F,F,F}, and split it into {T,T,T,T,F} and {T,F,F}

Entropy의 계산 편의성을 위해 파이썬 프로그램 코드로 계산과정을 작성하였습니다.

a. What are the entropy values of {T,T,T,T,T,F,F,F}, {T,T,T,T,F}, and {T,F,F}, respectively?

```python
import math

def prob(data):
    count = 0
    for i in range(len(data)):
        if data[i]==1:
            count+=1
    return count/len(data),1-count/len(data)

def entropy(prob1,prob2):
    return (-1)*prob1*math.log2(prob1)+(-1)*prob2*math.log2(prob2)

data = [1,1,1,1,1,0,0,0]
split1 = [1,1,1,1,0]
split2 = [1,0,0]

data_prob1,data_prob2 = prob(data)
split1_prob1,split1_prob2 = prob(split1)
split2_prob1,split2_prob2 = prob(split2)

data_entropy = entropy(data_prob1,data_prob2)
split1_entropy = entropy(split1_prob1,split1_prob2)
split2_entropy = entropy(split2_prob1,split2_prob2)
```

이런식으로 T -> 1 / F -> 0으로 나타내어 T와 F의 확률을 각각 구해서 entropy를 계산해줬습니다. split된 2개의 부분집합들에 대해서도 T와 F의 확률을 각각 구해서 entropy를 계산해줬습니다.

```python
print("original data entropy:", round(data_entropy,3))
print("split1 data entropy:", round(split1_entropy,3))
print("split2 data entropy:", round(split2_entropy,3))
```

```
original data entropy: 0.954
split1 data entropy: 0.722
split2 data entropy: 0.918
```

표현의 간결함을 위해서 소수점 3번째 자리까지만 나타내었습니다.
그 결과, 기존 집합인 {T,T,T,T,T,F,F,F}의 엔트로피는 0.954가 나왔고, split된 후의 부분집합인 {T,T,T,T,F}의 엔트로피는 0.722가 나왔고, split된 후의 부분집합인 {T,F,F}의 엔트로피는 0.918이 나왔습니다.

b. What is the average entropy of {T,T,T,T,F} and {T,F,F}?
이제 average entropy를 구해야 하는데 집합 2개로 split되었으므로,
average entropy = (split1 집합의 데이터 개수/전체 집합의 데이터 개수)*split1 집합의 entropy + (split2 집합의 데이터 개수/전체 집합의 데이터 개수)*split2 집합의 entropy로 구할 수 있습니다.

```
avg_splitentropy = (len(split1)/len(data))*split1_entropy + (len(split2)/len(data))*split2_entropy
print("average of split data entropy:", round(avg_splitentropy,3))
```
```
average of split data entropy: 0.796
```
따라서, average entropy of {T,T,T,T,F} and {T,F,F}는 0.796이 나왔습니다.

c. What is the gain of the split?

the gain of split = split되기 전 원래 집합의 entropy – split된 부분집합들의 average entropy이라서 다음과 같이 계산해줬습니다.

```
gain = data_entropy - avg_splitentropy
print("gain of the split:", round(gain,3))
```
```
gain of the split: 0.159
```
따라서, gain of split은 0.159가 나왔습니다.

2. Build a decision tree by ID3

2. Build a decision tree by ID3

	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	No
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

주어진 데이터에 대해서 ID3방법을 이용한 Decision Tree를 생성하는 문제여서 일일이 수동으로 계산하기 보다는 파이썬 코드를 통해 구현하는 방법이 더 간단하다고 판단되어 파이썬으로 ID3 생성 프로그램을 작성했습니다. 샘플 코드는 다음과 같습니다.

우선 기본적으로 위의 표에 존재하는 14개의 데이터를 리스트와 튜플을 통해서 구현해줬고, prob 함수와 entropy 함수를 통해 계산하여 DT라는 Decision Tree를 만들어내는 함수를 작성해줬고, checkDT로 Decision Tree가 정상적으로 나오는지 확인해줬습니다.

```python
import math

def prob(data):
    count = 0
    if len(data)==0:
        return 0, 0
    for i in range(len(data)):
        if data[i][4]=='Yes':
            count+=1
    return count/len(data),1-count/len(data)

def entropy(prob1,prob2):
    if prob1 == 0:
        logprob1 = 0
    else:
        logprob1 = math.log2(prob1)
    if prob2 == 0:
        logprob2 = 0
    else:
        logprob2 = math.log2(prob2)
    return (-1)*prob1*logprob1+(-1)*prob2*logprob2
```

```python
level = 0
decision_tree = []
data = [
    ('Sunny', 'Hot',    'High',   'Weak',   'No'),
    ('Sunny', 'Hot',    'High',   'Strong', 'No'),
    ('Overcast', 'Hot', 'High',   'Weak',   'Yes'),
    ('Rain', 'Mild',    'High',   'Weak',   'Yes'),
    ('Rain', 'Cool',    'Normal', 'Weak',   'Yes'),
    ('Rain', 'Cool',    'Normal', 'Strong', 'No'),
    ('Overcast', 'Cool','Normal', 'Strong', 'No'),
    ('Sunny', 'Mild',   'High',   'Weak',   'No'),
    ('Sunny', 'Cool',   'Normal', 'Weak',   'Yes'),
    ('Rain', 'Mild',    'Normal', 'Weak',   'Yes'),
    ('Sunny', 'Mild',   'Normal', 'Strong', 'Yes'),
    ('Overcast', 'Mild','High',   'Strong', 'No'),
    ('Overcast', 'Hot', 'Normal', 'Weak',   'Yes'),
    ('Rain', 'Mild',    'High',   'Strong', 'No')
]
DT(data, decision_tree, level,"")
checkDT(decision_tree)
```

level은 Decision Tree의 계층을 나타내는 변수로 사용해줬고, decision_tree 리스트에 값을 담아 tree를 표현했습니다.

```python
def DT(data, decision_tree, level, bestname):
    if len(data) == 0:
        del decision_tree[-1]
        return
    print("level : ", level)
    gainlist = []
    endcheck = 0
    for i in range(len(data)):
        if data[i][4]=='Yes':
            endcheck+=1
    if endcheck == len(data):
        print("Yes leaf node")
        decision_tree.append(("Yes", level))
        return
    elif endcheck == 0:
        print("No leaf node")
        decision_tree.append(("No", level))
        return
    if bestname != "Outlook":
        split_outlook1 = []
        split_outlook2 = []
        split_outlook3 = []

        for i in range(len(data)):
            if data[i][0]=='Sunny':
                split_outlook1.append(data[i])
            elif data[i][0]=='Overcast':
                split_outlook2.append(data[i])
            elif data[i][0]=='Rain':
                split_outlook3.append(data[i])

        outlook_prob1, outlook_prob2 = prob(data)
        sunny_prob1, sunny_prob2 = prob(split_outlook1)
        overcast_prob1, overcast_prob2 = prob(split_outlook2)
        rain_prob1, rain_prob2 = prob(split_outlook3)
        outlook_entropy = entropy(outlook_prob1, outlook_prob2)
        sunny_entropy = entropy(sunny_prob1, sunny_prob2)
        overcast_entropy = entropy(overcast_prob1, overcast_prob2)
        rain_entropy = entropy(rain_prob1, rain_prob2)
        # print("Outlook level : ", level)
        # print("original data entropy:", round(outlook_entropy,3))
        # print("split1 data entropy:", round(sunny_entropy,3))
        # print("split2 data entropy:", round(overcast_entropy,3))
        # print("split3 data entropy:", round(rain_entropy,3))

        avg_outlook_split_entropy = (len(split_outlook1)/len(data))*sunny_entropy+(len(split_outlook2)/len(data))*overcast_entropy+(len(split_outlook3)/len(data))*rain_entropy
        print("average of outlook split entropy:", round(avg_outlook_split_entropy,3))

        outlook_gain = outlook_entropy - avg_outlook_split_entropy
        print("gain of the outlook split:", round(outlook_gain,3))
        gainlist.append({"Outlook":outlook_gain})

    if bestname != "Temp":
        split_temp1 = []
        split_temp2 = []
        split_temp3 = []
```

이런식으로 처음 받은 data 값이 없다면 바로 리턴해줬고, data 값이 모두 yes거나 no면 split하지 않아도 되므로 그대로 리턴할 수 있게 해줬습니다. Outlook 특성의 값(Sunny, Overcast, Rain)에 대해서 split한 후의 엔트로피를 구한 후, 그들의 평균 엔트로피를 split되기 전의 엔트로피에서 빼서 Gain을 구해줬습니다. 이 과정을 Outlook, Temp, Humidity, Wind 4개의 속성에 대해서 똑같이 수행해줬습니다. 그 후 4개의 속성에 대한 Gain의 최댓값을 구해 해당 속성의 이름을 bestname으로 할당하고 bestname을 기준으로 split하여 같은 과정을 반복해줬습니다. 이 과정을 재귀함수로 구현했고 코드는 다음과 같습니다.

```
    bestattr = max(gainlist, key=lambda x: list(x.values())[0])
    bestname = list(bestattr.keys())[0]
    print(bestname)
    decision_tree.append((bestname, level))
    if bestname == "Outlook":
        decision_tree.append(("Sunny", level+1))
        DT(split_outlook1, decision_tree, level+2, bestname)
        decision_tree.append(("Overcast", level+1))
        DT(split_outlook2, decision_tree, level+2, bestname)
        decision_tree.append(("Rain", level+1))
        DT(split_outlook3, decision_tree, level+2, bestname)
    elif bestname == "Temp":
        decision_tree.append(("Hot", level+1))
        DT(split_temp1, decision_tree, level+2, bestname)
        decision_tree.append(("Mild", level+1))
        DT(split_temp2, decision_tree, level+2, bestname)
        decision_tree.append(("Cool", level+1))
        DT(split_temp3, decision_tree, level+2, bestname)
    elif bestname == "Humidity":
        decision_tree.append(("High", level+1))
        DT(split_humidity1, decision_tree, level+2, bestname)
        decision_tree.append(("Normal", level+1))
        DT(split_humidity2, decision_tree, level+2, bestname)
    elif bestname == "Wind":
        decision_tree.append(("Weak", level+1))
        DT(split_wind1, decision_tree, level+2, bestname)
        decision_tree.append(("Strong", level+1))
        DT(split_wind2, decision_tree, level+2, bestname)
    return
```

data가 split된 각각의 데이터에 대해서 재귀함수를 돌도록 실행해줬습니다. 이렇게 재귀함수를 돌면서 Yes leaf node나 No leaf node를 만날 때까지 즉, pure한 노드가 나타날 때까지 코드가 실행됩니다. 이 모든 과정이 DT 함수를 통해 실행되고, decision_tree.append()를 통해 속성의 값과 level(계층)이 저장됩니다.

```
def checkDT(decision_tree):
    strs = ""
    count = 0
    decision_tree.sort(key=lambda x: x[1])

    for i in range(len(decision_tree)):
        if decision_tree[i][1] == count:
            strs = strs + decision_tree[i][0] + " "
        else:
            print(strs)
            count+=1
            strs = decision_tree[i][0] + " "
    print(strs)
```

그리고 checkDT 함수를 통해서 계층별로 노드들을 출력하여 Decision Tree를 확인할 수 있도록 하였습니다.

코드 실행의 결과는 다음과 같습니다.

```
level :  0
average of outlook split entropy: 0.979
gain of the outlook split: 0.021
average of temp split entropy: 1.0
gain of the temp split: 0.0
average of humidity split entropy: 0.863
gain of the humidity split: 0.137
average of wind split entropy: 0.742
gain of the wind split: 0.258
Wind
level :  2
average of outlook split entropy: 0.344
gain of the outlook split: 0.467
average of temp split entropy: 0.689
gain of the temp split: 0.123
average of humidity split entropy: 0.5
gain of the humidity split: 0.311
Outlook
level :  4
average of temp split entropy: -0.0
gain of the temp split: 0.918
average of humidity split entropy: -0.0
gain of the humidity split: 0.918
average of wind split entropy: 0.918
gain of the wind split: 0.0
Temp
level :  6
No leaf node
level :  6
No leaf node
level :  6
Yes leaf node
level :  4
Yes leaf node
level :  4
Yes leaf node
```

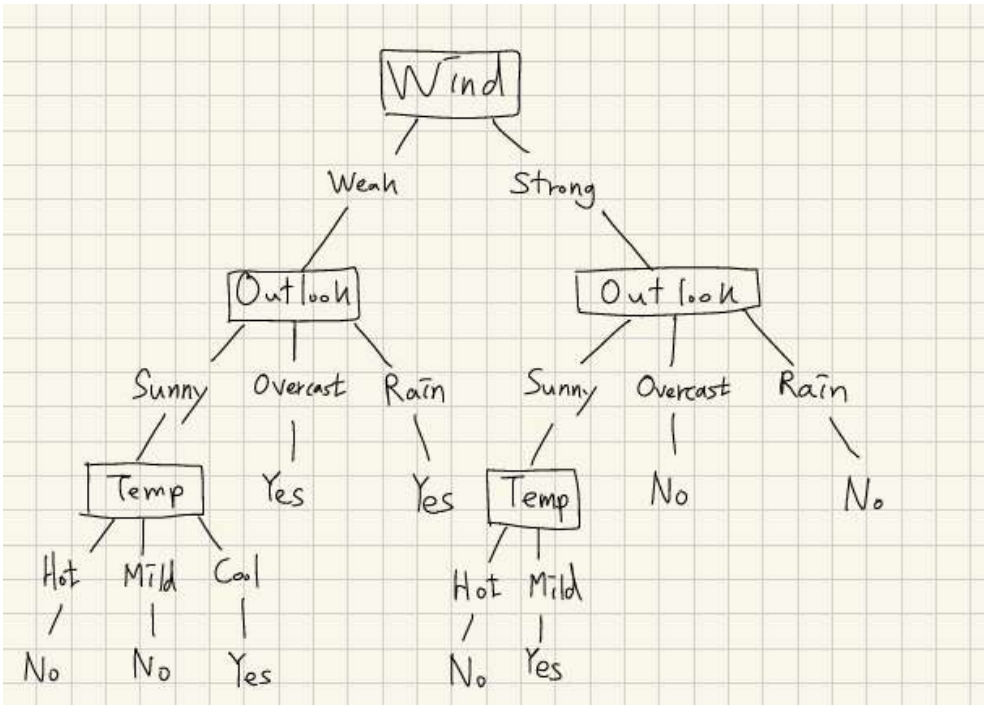
이런 식으로 처음에 level = 0이므로, 전체 데이터에 대해서 Gain을 계산해주고 Gain값이 가장 큰 Wind 속성을 기준으로 split되어 level = 2로 진입합니다.(level 2로 설정해준 이유는 level 1에서는 Wind의 속성의 값(Weak, Strong)이 출력되도록 하기 위함입니다.)
level 2에서 또 다시 같은 과정을 통해 Outlook으로 split했을 때 Gain이 가장 크다는 것을 발견하고, Outlook 속성을 기준으로 split되어 level = 4로 진입합니다.
level 4에서는 Temp와 Humidity의 Gain이 똑같으므로 Temp를 기준으로 split되어 level = 6으로 진입합니다.
level 6에서는 Weak(Wind)-Sunny(Outlook)를 만족하는 데이터는 3개 밖에 없는데 Hot, Mild, Cool 1개씩이라서 각각 split되면 모두 Yes 혹은 No를 만족시키는 리프노드가 됩니다. 따라서, level:6 No/Yes leaf node가 3번 출력된 것입니다. 그 후 Overcast(level 4)는 모두 Yes이므로 Yes leaf node가 출력되었고, Rain도 마찬가지로 모두 Yes라서 Yes leaf node가 출력되었습니다. 이 과정을 Wind = Strong인 경우에도 똑같은 과정을 통해 진행되어 Decision Tree를 생성하였습니다.

결론적으로, checkDT 함수를 통해 Decision Tree를 출력해보면 결과는 다음과 같습니다.

```
Wind
Weak Strong
Outlook Outlook
Sunny Overcast Rain Sunny Overcast Rain
Temp Yes Yes Temp No No
Hot Mild Cool Hot Mild
No No Yes No Yes
```

이 결과를 최종적으로 그림으로 다시 나타내보겠습니다.

<Decision Tree by ID3>



최종적으로 이런 형태의 Decision Tree가 생성됩니다. 다만, Strong -> Sunny -> Cool의 branch가 없는 이유는 Strong -> Sunny를 만족시키는 데이터는 (Sunny, Hot, High, Strong, No), (Sunny, Mild, Normal, Strong, Yes) 이렇게 2개 밖에 없습니다. 즉, 이 조건을 만족하는 데이터의 Temp 속성 값이 Cool을 제외한 Hot, Mild 2개만 존재하기 때문에 Decision Tree의 branch 역시 Hot, Mild 2개만 표현됩니다.