지능정보수학 과제

201910810

안성찬

목차

1. entropy 함수에 추가한 부분입니다.

```
entropy = 0      # 결괏값으로 반환할 entropy 변수를 정의하였습니다.
for count in counts: # target_col 안에 unique한 column의 개수인 counts를
                     for문으로 돌면서 확률(_p)과 엔트로피(entropy)를 구합니다.
    _p = count / sum(counts) # unique한 값의 확률, P(t=i)를 구합니다.
                             unique한 값의 개수에 전체 개수를 나누어 _p를 구합니다
    entropy += -_p * np.log2(_p) # -Σ_{i∈level set(t)} (P(t=i) x log2(P(t=i)))를 구합니다. -를 붙이고
                                   각각의 확률과 log2를 씌운 확률을 곱하여 더합니다.
                                   이렇게 label에 대한 entropy, H(t,D)가 구해졌습니다.
```

전체 코드:

```
entropy =0
for count in counts:
    _p= count / sum( counts)
    entropy += -_p * np. log2(_p)
```

2. Info_Gain 함수에 추가한 부분입니다.

Weighted_Entropy = 0 # 결괏값으로 반환할 Weighted_Entropy 변수를 정의하였습니다.

for val, count in zip(vals, counts): # Weighted_Entropy를 구하기 위해서 개별 피처를 unique한 값은 vals와 그 값의 개수인 counts를 zip함수로 묶어 for문을 돕니다.

$rem(d,D) = \sum_{i \in levels(d)} \underbrace{\frac{|D_{d=i}|}{|D|}}_{\text{weighting}} \times \underbrace{H(t, D_{d=i})}_{\text{entropy of Partition } D_{d=i}}$ 를 구현합니다.

weighting = count / sum(counts) # weighting을 구하기 위해서 개별 unique한 값으의 개수인 count에 전체 개수인 sum(counts)를 나누어서 $\frac{|D_{d=i}|}{|D|}$, weighting을 구합니다.

_target = data.where(data[split_attribute_name] == val).dropna()[target_name]
# 개별 피처의 값이 val인 것만 필터링을 해서 dropna() 함수로 달랑값을 제거한 다음 필터링된 [target_name] ("class") 피처만 Series로 반환합니다.

target_elements, target_counts = np.unique(_target, return_counts = True)
unique함수에 return_counts=True 인자를 넣어서 개별 값과 개수를 구합니다.

entropy_of_partition = 0 # 개별 피처 필터링을 했을 때의 target_name의 엔트로피를 저장하기 위해 entropy_of_Partition 변수를 정의합니다.

for target_count in target_counts: # entropy of Partition을 구하기 위해 for문을 한번 더 돌립니다.

_p = target_count / sum(target_counts) # 개별 피처에 대한 entropy인

entropy_of_partition += -_p * np.log2(_p) entropy of Partition을 구합니다.

Weighted_Entropy += weighting * entropy_of_partition

weighting과 entropy_of_Partition을 곱하여 속성별 엔트로피를 구하고 모두 더하여 Weighted_Entropy를 구합니다.

Information_Gain = total_entropy - Weighted_Entropy

total_entropy와 Weighted_Entropy의 차로 분할 전 Entropy와 분할 후 Entropy의 차이인 Information_Gain을 구합니다.

전체 코드:

```
Weighted_Entropy = 0
for val, count in zip(vals, counts):
    weighting = count / sum(counts)
    _target = data.where(data[split_attribute_name] == val).dropna()[target_name]
    target_elements, target_counts = np.unique(_target, return_counts = True)
    entropy_of_partition = 0
    for target_count in target_counts:
        _p = target_count / sum(target_counts)
        entropy_of_partition += -_p * np.log2(_p)
        Weighted_Entropy += weighting * entropy_of_partition
Information_Gain = total_entropy - Weighted_Entropy
```

3. 실행 결과, 고찰 및 의미

3.1. 실행 결과

```
{'legs': {0: {'fins': {0.0: {'toothed': {0.0: 7.0, 1.0: 3.0}},
                       1.0: {'eggs': {0.0: 1.0, 1.0: 4.0}}}},
          2: {'hair': {0.0: 2.0, 1.0: 1.0}},
          4: {'hair': {0.0: {'toothed': {0.0: 7.0, 1.0: 5.0}}, 1.0: 1.0}},
          6: {'aquatic': {0.0: 6.0, 1.0: 7.0}},
          8: 7.0}}
The prediction accuracy is:  85.71428571428571 %
Press any key to continue . . .
```

3.2. 고찰

Decision_tree_HW.py는 동물원의 동물들을 'legs', 'fins', 'toothed' 등의 특징으로 분류해내는 의사결정 트리 모델을 파이썬으로 구현한 코드이다. C5.0, CART, CHAID 등의 많은 의사 결정 트리 알고리즘이 있지만 여기서는 ID3 알고리즘이 사용되었다. ID3 알고리즘은 Entropy를 큰 값부터 분할하며 Entropy를 작게 하는 방향으로 가지를 뻗어 나가며 데이터를 분할한다. 코드에서는 재귀 방식으로 분할이 이루어졌다. ID3 알고리즘은 데이터가 범주형일 때 유용하다. zoo.csv 데이터셋이 'legs'를 제외하면 1과 0으로 이루어진 범주형 자료였기 때문에 ID3 알고리즘은 적합한 알고리즘이었다.

실행 결과를 보면, 첫번째 Best feature인 'legs'로 먼저 분기가 되어 0, 2, 4, 6, 8으로 먼저 분할이 된다. 그 후에 ID3 함수의 재귀를 통해 분할 후의 속성에 따른 Best feature으로 다시 분할이 되는 것을 볼 수 있다. 실행 결과의 첫번째 줄인 'legs', 'fins', 'toothed' 순서로 분기가 되는 것을 Jupyter notebook에서 실제 데이터셋으로 확인해 보면 다음과 같다.

```
dataset.where(dataset['legs']==0).where(dataset['fins']==0).where(dataset['toothed'] == 0).dropna()
```

	hair	feathers	eggs	milk	airbone	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	class
13	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0
77	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	7.0
81	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0
99	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0

['toothed']까지 분할이 이루어졌을 때 class의 값은 모두 7으로 ID3 알고리즘의 첫번째 조건문으로 7이 반환이 된다.

3.3. 의미

의사결정 트리는 분류와 같은 의사결정을 할 때 나무와 같이 가지치기를 함으로써 분류하는 방법이다. 코드에서는 Entropy를 이용하는 ID3 알고리즘이 사용되었다. 의사결정 트리는 범주형 데이터일 때 사용이 유리하며, 직관적으로 이해하기 쉽다는 장점이 있다. 다만 의사결정 트리는 과적합이 되기 쉽다는 문제가 있다. Training data에 지나치게 학습이 되었을 경우에는 training data의 어떤 특징들을 모든 데이터가 가지고 있는 일반적인 특성으로 착각하여 과적합이 일어날 수 있다. 의사결정 트리의 과적합에 대응하기 위한 방법으로는 가지치기가 있다. 사전가지치기 혹은 사후가지치기로 나누어진다. 사전가지치기는 분할 전 미리 예측하여 일반화 성능을 향상시킬 수 없다면 해당 노드를 터미널 노드로 만드는 방법이고 사후 가지치기는 훈련세트를 통해 의사결정 트리를 만든 후 일반화 성능을 향상시킬 수 있다면 하위 트리를 터미널 노드로 바꾸는 방법이다.

실행 결과가 의미하는 것은 실행 결과에 따른 feature의 순서대로, 즉 엔트로피가 낮아지는 순서대로 새로운 데이터에 대하여 분류를 해 나갔을 때 가장 빠르게 target_attribute를 구할 수 있다는 것이다. 또한 대략 85%의 The prediction accuracy가 출력이 되었는데, 이는 testing_data로 분류한 값과 실제 class 값을 비교했을 때의 정확도이다.