보고서

Decision Tree : ID3 Algorithm

제출일	2023.12.23	전 공	휴먼지능정보공학전공
과 목	인공지능개론	학 번	202115047
담당교수님	이지항 교수님	이 름	김희재

목차

1. 서론

1.1 Decision Tree

Decision Tree 는 분류와 회귀 작업 등이 가능한 머신 러닝 방법으로, 질문을 통해 간단한 결정을 내려 데이터셋을 학습한다. 매우 복잡한 데이터 세트도 학습할 수 있는 아주 강력한 알고리즘이다. Decision Tree 맨 위에 있는 루트 노드에서 시작한다. 트리의 루트 노드 아래의 각 노드는 결정을 좌우하는 질문으로 구성되어 있고, interior node 라고 불린다. 이 질문들의 결과에 따라 데이터가 분할되어, 맨 마지막 leaf 노드에는 위 root 노드와 interior 노드들의 결과인, 실험자가 원하는 결과만이 남아있게 될 것이다. 결국 Decision Tree 의 목적은 데이터셋을 잘, 그리고 자원을 아끼며 효율적으로 분류하는 데에 있다.

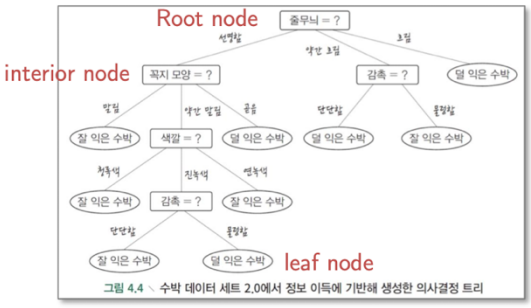


그림 4.4 ㅣ 수박 데이터 세트 2.0에서 정보 이득에 기반해 생성한 의사결정 트리

자원을 아끼며 효율적으로 데이터셋을 학습했다는 것은 무엇일까? 이는 트리의 depth 와 관련이 있다. 상식적으로 생각해본다면 당연히 적은 질문을 통해 학습의 결과를 얻는 것이 메모리와 시간 등의 자원적인 관점에서 매우 효율적인 방법일 것이다. 그러므로 root node 부터의 dpeth 를 얕게 만들 수 있는 적절한 질문 선택이 매우 중요하다. 즉, depth 를 얕게 만들 수 있는 질문 찾기는 곧 어떤 질문이 데이터셋을 잘 나눌 수 있는 확률이 가장 높은지와 같은 문제이고 있는 불순도(purity)를 측정하여 entropy 를 계산함으로써 수학적으로 측정할 수 있다. 확률이 큰 사건은 늘상 일어나는 사건, 특별하지 않은 사건으로 볼 수 있고 이는 곧 추측하려면 적은 정보가 필요함을 의미한다. 또 확률이 작은 사건은 잘 일어나지 않는 사건, 특별한 사건으로 볼 수 있고 이는 추측하려면 많은 정보가 필요하다 볼 수 있다. 확률이 큰 사건은 entropy 가 작고, 확률이 작은 사건은 entropy 가 큰 것이다. Entropy 값은 0 과 1 사이를 가진다. 데이터셋 안에 여러 사건들이 섞여 있어 불순도가 높은 상태라면 1 에 가까울 것이고, 순도가 높은 상태라면 0 에 가까울 것이다. 이렇게 수치화 된 데이터의 순도, Entropy 로 그럼 어느 질문이 데이터의 순도를 가장 잘 나누는지도 Entropy 들의 사칙연산으로 구할 수 있고 이를 Information Gain 이라고 한다. 결국 질문마다의 Information Gain 을 구해 가장 큰걸 선택하면 dept 가 얕은 트리를 구축할 수 있을 것이다. 오른쪽은 교수님의 강의 자료에 있는 Entropy 와 Information Gain 을 구하는 수식이다.

Step 1: Calculate the entropy for the label in the dataset

$$H(t,\mathcal{D}) = -\sum_{l \in levels(t)} (P(t=l) \times log_2(P(t=l)))$$

Step 2: Calculate the remainder for each feature

$$rem(d,\mathcal{D}) = \sum_{l \in levels(d)} \underbrace{\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|}}_{\text{weighting}} \times \underbrace{H(t,\mathcal{D}_{d=l})}_{\text{entropy of partition } \mathcal{D}_{d=l}}$$

dataset

전체 데이터에서 l 라벨을 가진 d feature 비율

Step 3: Calculate the information gain for all features

$$IG(d,\mathcal{D}) = H(t,\mathcal{D}) - rem(d,\mathcal{D})$$

1.2 ID3 Algorithm

ID3 알고리즘은 Decision Tree 를 만드는 알고리즘 중 하나로 재귀 알고리즘이다. 교수님께서 제공해주신 코드는 ID3 알고리즘으로 이루어져 있다. ID3 알고리즘은 주어진 데이터와 일치하는 가장 얕은 트리를 만들려고 시도한다. Root node 에서부터 시작하여 leaf node 까지 진행하는 깊이 우선적인 방식으로 구축된다.

ID3 알고리즘은 테스트할 최적의 첫번째 질문을 선택하는 것으로 시작한다. Information Gain 의 수치가 가장 큰 질문을 선택하면 된다. 그런 다음 root 노드가 트리에 추가되고 선택한 테스트 feature 로 레이블이 지정된다. 그 이후 또 훈련 데이터셋이 다시 분할되고, 각 분기에 대해 추가적인 가지가 성장한다. 이 과정은 마지막 leaf 노드가 동일한 분류 값을 갖도록 반복된다.

본 알고리즘의 장점은 가장 빠르고 짧은 나무를 만든다는 것이다. Understandable 한 prediction 규칙들이 training 데이터로부터 생성되고 leaf node 를 찾음으로써 test 데이터를 다 자를 수 있으므로 test 횟수도 줄어드는 장점이 있다. 하지만 데이터의 양이 적었을 경우 overfitting 이 될 위험이 있고 결정을 내리기 위해 한번에 하나의 attribute 만 test 되는 단점이 있다. 또한 카테고리형이 아닌 연속형 데이터를 분류할 때에는 많은 트리가 필요하기에 자원이 많이 낭비될 수도 있다.

이제 본론에서 ID3 알고리즘으로 구성된 코드를 자세히 보도록 하겠다.

2. 본론

2.1.1 zoo.csv 데이터

zoo

aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	1
antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	1
boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
buffalo	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
calf	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	1
carp	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	4
catfish	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
cavy	1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	1
cheetah	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
chicken	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	2
chub	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
clam	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	7
crab	0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	7
crayfish	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	7
crow	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	2
deer	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
dogfish	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	4
dolphin	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1

본 과제에 제공된 데이터를 열어서 관찰해보았다.처음에는 숫자들로 구성되어 있길래 one-hot encoding 이 되어 있는건가? 라는 생각을 하였는데 숫자가 0,1 외에도 다양하게 구성되어 있고 숫자가 겹쳐서 구성되어 있는 걸 보아 그런 건 아니고 그냥 수치화된 데이터임을 알았다. 동물들의 특징에 대한 정보를 담고 있는 것 같고, 각 행은 한 종류의 동물에 대한 정보를, 각 열은 다양한 특징을 나타내고 있는 것 같다. 수치화 되어 있으므로 이 데이터로 모델을 학습시키는 데에는 전혀 지장이 없어보인다.

2.1.2 기존 코드 전반의 이해

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 25 20:29:26 2020

@author: jeehang

Ackowledgement: https://www.python-course.eu/Decision_Trees.php
"""

"""
Make the imports of python packages needed
"""
import pandas as pd
import numpy as np
from pprint import pprint
```

우선 필요한 라이브러리들을 가져왔다. Pandas 는 Dataframe 이라는 자료구조를 제공해주는 라이브러리이고, numpy 는 다양한 수학 연산들을, pprint 는 데이터 구조를 예쁘게 출력하기 위해 사용된다.

```
# 데이터셋을 불러온 후 feature랑 target dataset을 정의한다.
dataset = pd.read_csv('zoo.csv',
                      names=['animal_name','hair','feathers','eggs','milk',
                     'airbone','aquatic','predator','toothed','backbone',
                     'breathes','venomous','fins','legs','tail','domestic','catsize','class',])


# 동물 이름은 데이터를 나누기 좋은 feature가 아니므로 제외한다.
dataset=dataset.drop('animal_name',axis=1)
```

우선 pd. read_csv 함수로 데이터를 불러오고 각 열에 대한 이름을 지정하였다. 위에서 데이터 파일만을 열어보았을 때는 몰랐는데 이렇게 보니 각 열이 descriptive features 에 대한 수치들이었고, 마지막 열은 정답 label 로 class 였다. 데이터를 불러온 후에는 animal_name 에 대한 feature 열을 데이터 분석하는 데에 필요하지 않기 때문에 삭제하였다.

```
# 데이터 집합의 엔트로피를 계산하는 함수
# target_col은 이 함수의 유일한 매개변수로 대상 열을 지정하는 변수이다.

def entropy(target_col):

    # np.unique 함수는 배열의 정렬된 고유 요소를 반환한다.
    # 추가적으로 return_counts를 아래와 같이 True로 하면 고유한 값 각각의 개수도 반환한다.
    elements, counts = np.unique(target_col, return_counts = True)

    # 과제1 : entropy 구하기

    # 기존 코드
    entropy = np.sum([(-counts[i] / np.sum(counts)) * np.log2(counts[i] / np.sum(counts)) for i in range(len(elements))])

    # 변경 코드 1
    # probabilities = counts / np.sum(counts)
    # entropy = -np.sum(probabilities * np.log2(probabilities), axis=0)

    # 변경 코드 2
    # probabilities = counts / np.sum(counts)
    # entropy = -np.einsum('i,i->', probabilities, np.log2(probabilities))

    return entropy
```

다음은 entropy 함수이다. 이 함수는 데이터 집합의 엔트로피를 계산하는 함수로 본 함수의 유일한 매개변수는 target_col 이다. 이는 대상 열을 지정하는 변수이다. np.unique 함수를 사용하여 각 고유한 값들을 elements 에 넣고, 그 각각의 개수들을 counts 변수에 넣은 걸 볼 수 있다. 이 함수를 처음 알게 되었는데, 최빈값을 구할 때 사용되어도 참 좋을 유용한 함수인 것 같다. 그리고 나서는 entropy 를 numpy 와 for 문을 이용하여 구한 이후 함수의 반환값으로 지정하였다. Entropy 에 관한 내용은 2.2 장에서 자세히 보도록 하겠다.

```
# 데이터 집합의 Information Gain을 계산하는 함수

def InfoGain(data,split_attribute_name,target_name="class"):

    """
    데이터셋의 Information Gain을 계산
    Information Gain 함수는 세 가지 매개변수를 사용한다:
    1. data = Information Gain을 계산할 데이터셋
    2. split_attribute_name = Information Gain을 계산할 attribute의 이름
    3. target_name = target의 이름. 기본값은 "class"이다.
    """

    # 전체 데이터셋의 entropy 계산
    total_entropy = entropy(data[target_name])

    ## 데이터셋의 entropy 계산

    # 위와 같이 각 attribute 에 대한 값 및 각각의 개수 계산
    vals, counts= np.unique(data[split_attribute_name],return_counts=True)

    # 과제2 : Weighted Entropy 계산 - 아래 코드를 자신만의 코드로 변경하고 결과를 확인해 봅시다.

    # 기존 코드
    Weighted_Entropy = np.sum([(counts[i] / np.sum(counts)) * entropy(data.where(data[split_attribute_name] == vals[i]).dropna()[target_name])
                               for i in range(len(vals))])

    # 변경 코드 1
    # Weighted_Entropy = np.sum([(count / np.sum(counts)) * entropy(data[data[split_attribute_name] == val][:][target_name]) \
                          # for val, count in zip(vals, counts)])

    # 변경 코드 2
    # Weighted_Entropy = np.sum((counts / np.sum(counts)) * np.array([entropy(data[data[split_attribute_name] == val][target_name]) \
                           # for val in vals]))

    # Information Gain 계산
    Information_Gain = total_entropy - Weighted_Entropy

    return Information_Gain
```

그 다음은 데이터 집합의 Information Gain 을 구하는 함수인 InfoGain 함수가 정의되어 있다. InfoGain 함수는 data, split_attribute_name, target_name 이렇게 세가지 매개변수를 사용한다. data 는 Information Gain 을 계산할 데이터셋이고 split_attribute_name 은 Information Gain 을 계산할 attribute 의 이름이다. 그리고 target_name 은 class 이다. 우선 전체 데이터셋의 entropy 를 위 entropy 함수로 계산한 것을 알 수 있고, vals, counts 도 위 함수와 같이 np.unique 를 사용하여 구한 것을 볼 수 있다. Information Gain 계산을 위해 Weighted Entropy 를 구하여 Total entropy 에서 이를 빼고 이를 함수의 반환 값으로 지정함으로 Information Gain 함수가 구성되어 있다.

```
"""
    ID3 알고리즘: 이 함수는 다섯 가지 매개변수를 사용한다.

    1. data = ID3 알고리즘이 실행되어야 하는 데이터 --> 첫 번째 실행에서는 전체 데이터셋과 동일하다.

    2. originaldata = 첫 번째 매개변수로 전달된 데이터셋이 비어 있을 경우 원본 데이터셋이 필요하며, 이를 통해 원본 데이터셋의 모드 타겟 피처 값을 계산한다.

    3. features = 데이터셋의 feature 공간입니다. 이는 tree 성장 process 중에 데이터셋에서 feature를 제거해야 하므로 재귀 호출에 필요하다.

    4. target_attribute_name = 대상 속성의 이름이다.

    5. parent_node_class = 특정 노드에 대한 부모 노드의 모드 target feature 값 (또는 class) 이다.
       이는 재귀 호출에서 사용되며, feature 공간에 더 이상 feature가 남아 있지 않은 상황에서 직접 부모 노드의 mode target feature 값을 반환하려는 경우 필요하다.
"""
def ID3(data, originaldata, features, target_attribute_name="class", parent_node_class=None):

    # 중단 조건을 정의
    # 이 중 하나라도 만족하면 leaf 노드를 반환한다.

    # 모든 target_values가 동일한 값을 가지면 해당 값을 반환한다.
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    # 데이터셋이 비어 있으면 원본 데이터셋에서 mode target feature 값을 반환한다.
    elif len(data) == 0:
        return np.unique(originaldata[target_attribute_name])[
            np.argmax(np.unique(originaldata[target_attribute_name], return_counts=True)[1])
        ]

    # feature 공간이 비어 있으면 직전 호출에서 저장된 부모 노드의 mode target feature 값을 반환한다.
    elif len(features) == 0:
        return parent_node_class

    # 위의 조건 중 어느 것도 만족하지 않으면 트리를 성장시킨다!
```

```
    else:
        # 현재 노드의 기본값을 설정 --> 현재 노드의 mode target feature 값
        parent_node_class = np.unique(data[target_attribute_name])[
            np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])
        ]

        # 데이터셋에서 가장 좋은 feature를 선택한다.
        item_values = [InfoGain(data, feature, target_attribute_name) for feature in features]

        # 데이터셋의 각 feature에 대한 Information Gain 값을 반환한다.
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]

        # 트리 구조를 생성
        # 첫 번째 실행에서 가장 큰 Information Gain을 가진 feature의 이름을 root로 지정
        tree = {best_feature: {}}

        # Information Gain 이 가장 큰 feature를 feature 공간에서 제거
        features = [i for i in features if i != best_feature]

        # root 노드 하위에 root 노드 feature의 각 가능한 값에 대한 가지를 성장시킨다.
        for value in np.unique(data[best_feature]):
            value = value

            # 데이터셋을 가장 큰 Information Gain을 가진 feature의 값으로 나누어 하위 데이터셋을 생성
            sub_data = data.where(data[best_feature] == value).dropna()

            # 이러한 하위 데이터셋 각각에 대해 새 매개변수로 ID3 알고리즘을 호출 --> 여기에서 재귀가 발생!
            subtree = ID3(sub_data, originaldata, features, target_attribute_name, parent_node_class)

            # 하위 데이터셋에서 성장한 서브 트리를 루트 노드 아래에 추가합니다.
            tree[best_feature][value] = subtree

        return tree
```

다음은 ID3 알고리즘에 대한 함수로 다섯 가지 매개변수를 사용한다. 첫번째 data 는 ID3 알고리즘이 실행되어야 하는 데이터로 첫 번째 실행에서는 전체 데이터셋과 동일하다. 두번째 originaldata 는 첫 번째 매개변수로 전달된 데이터셋이 비어 있을 경우 원본 데이터셋이 필요하기 때문에 이를 통해 원본 데이터셋의 모드 타겟 피처 값을 계산한다. 세번째 feature 는 데이터셋의

feature 공간이다. 이는 tree 성장 process 중에 데이터셋에서 feature 를 제거해야 하므로 재귀 호출에 필요하다. 네번째 target_attribute_name 는 대상 속성의 이름이고, parent_node_class 는 특정 노드에 대한 부모 노드의 모드 target feature 값 (또는 class) 이다. Parent_node_class 변수는 재귀 호출에서 사용되며, feature 공간에 더 이상 feature 가 남아 있지 않은 상황에서 직접 부모 노드의 mode target feature 값을 반환하려는 경우 필요하다.

그래서 함수 내부를 보면 if 문으로 구성되어 있는데, if문의 else가 나오기 전까지는 중단 조건을 정의한 것이다. 첫번째 모든 데이터가 동일한 대상 속성 값을 갖는 경우 해당 값을 반환하여 leaf 노드를 생성하도록 하고, 두번째 데이터셋이 비어 있는 경우 원본 데이터셋에서 mode 대상 속성 값을 반환하도록 하고, 마지막으로 특성 공간이 비어 있는 경우 이전 호출에서 저장된 부모 노드의 mode 대상 속성 값을 반환하도록 중단 조건이 구성되어 있다. Else 조건에 걸렸을 경우 ID3 알고리즘에 따라 트리를 성장시킨다. 현재 노드의 모드 대상 속성 값을 저장하고, InfoGain 함수와 for 문을 이용해 dataset 의 각 특성에 대한 Information Gain 값을 계산하도록 한다. 그리고 가장 큰 Information Gain 값을 갖는 특성을 선택한 이후 트리를 성장시키고, 그 밑에 재귀호출을 진행하여 중단 조건을 맞이하기 전까지 본 함수가 재귀적으로 순환하도록 코드가 구성되어 있다.

```
def predict(query, tree, default=1):

    # 1.
    for key in list(query.keys()):
        if key in list(tree.keys()):
            # 2.
            try:
                result = tree[key][query[key]]
            except:
                return default

            # 3.
            result = tree[key][query[key]]
            # 4.
            if isinstance(result, dict):
                return predict(query, result)
            else:
                return result
```

그 다음은 Predict, 예측 함수이다. 이는 새로운, 보지 않은 query instance 의 예측을 수행하는 함수로 query 와 tree 두개의 매개변수를 갖는다. 교수님이 제공해주신 주석과 함께 코드를 보면, 우선 query 의 feature 의 값과 tree 의 feature 의 값을 비교하여, 루트 노드부터 query 와 일치하는지 확인하며 일치하는 경우 트리를 내려가도록 한다. 내려간 끝에 leaf node 를 찾으면 leaf 를 반환하고 그것이 예측의 결과가 된다. 만약 Leaf node 가 아닌 다른 node 를 찾으면 query 에서 그 node 와 일치하는 기능을 찾아 아래로 내려가도록 한다. 이는 재귀적인 과정으로 구성되어 있다.

```python
def train_test_split(dataset):
    # 훈련 데이터와 테스트 데이터로 데이터셋을 나눈다.
    # 주어진 데이터셋을 80:20 비율로 나누어 훈련 데이터와 테스트 데이터로 사용한다.
    # reset_index(drop=True)를 통해 새로운 인덱스를 생성하고 이를 0부터 다시 라벨링한다.
    training_data = dataset.iloc[:80].reset_index(drop=True)
    testing_data = dataset.iloc[80:].reset_index(drop=True)

    return training_data, testing_data

# train_test_split 함수를 사용하여 훈련 데이터와 테스트 데이터를 얻는다.
training_data = train_test_split(dataset)[0]
testing_data = train_test_split(dataset)[1]
```

위 코드는 train 데이터와 test 데이터를 나누는 함수이다.

```python
def test(data, tree):
    # 원본 데이터셋에서 대상 feature 열을 단순히 제거하여 새로운 쿼리 instance를 생성하고
    # 이를 dictionary로 변환
    queries = data.iloc[:, :-1].to_dict(orient="records")

    # 예측된 결과를 저장할 빈 DataFrame을 생성
    predicted = pd.DataFrame(columns=["predicted"])

    # 예측 정확도를 계산
    for i in range(len(data)):
        # 각 쿼리에 대한 예측을 계산하고 결과를 predicted DataFrame에 저장
        predicted.loc[i, "predicted"] = predict(queries[i], tree, 1.0)

    # 예측 정확도를 출력
    print('예측 정확도는: ', (np.sum(predicted["predicted"] == data["class"]) / len(data)) * 100, '%')
```

Test 함수에서는 predict 함수를 호출하여 test 한 이후 예측 정확도를 계산하고 출력한다.

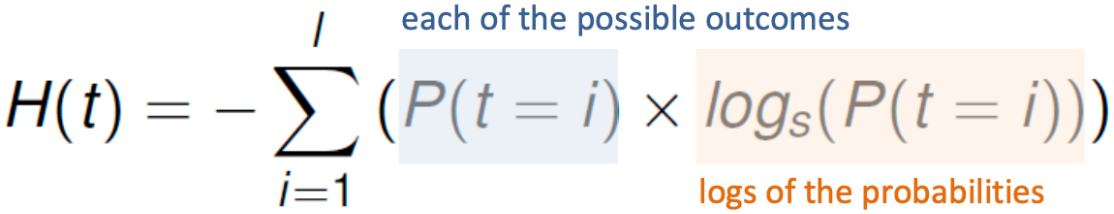
```python
# Train
# Tree와 최종 prediction accuracy 출력

def build_decision_tree():

    tree = ID3(training_data,training_data,training_data.columns[:-1])

    pprint(tree)

    test(testing_data,tree)

if __name__ == '__main__':
    build_decision_tree()
```

```
{'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}}
예측 정확도는:  85.71428571428571 %
```

Build_decision_tree 함수에서는 ID3 함수를 호출하여 tree 를 만든 이후 pprint 함수로 이를 예쁘게 출력하고 test 함수로 예측 후 정확도 출력을 진행하여 모든 코드가 잘 돌아가도록 한다.

2.2 Entropy

2.2.1 기존 코드




```
# 기존 코드
entropy = np.sum([(-counts[i] / np.sum(counts)) * np.log2(counts[i] /
np.sum(counts)) for i in range(len(elements))])
```

기존코드는 위처럼 작성되어 있었다. 주어진 데이터셋의 엔트로피를 계산하는 코드이다. 수업에서 배운 식과 정확히 일치하는 걸 자세히 들여다 보면 알 수 있다. 우선 시그마를 뜻하는 np.sum 이 for 문과 함께 존재한다. -가 붙어있고, 기존 확률과 log 가 붙은 확률을 곱하는 식을 코드에서도 명확히 볼 수 있다.

2.2.2 변경 코드 및 설명

위 코드를 아래와 같이 두가지 버전으로 변경하여 보았다.

```
# 변경 코드 1
probabilities = counts / np.sum(counts)
entropy = -np.sum(probabilities * np.log2(probabilities), axis=0)
```

이 코드는 확률을 계산하는 부분을 밖으로 빼고 np.sum 에 넣은 코드이다. 여기서 for 문이 없어도 코드가 동일하게 돌아가는 이유는 axis=0 을 설정해두면 첫번째 axis 를 따라 합계를 취합하도록 numpy.sum 함수가 구성되어 있기 때문이다.

```
# 변경 코드 2
probabilities = counts / np.sum(counts)
entropy = -np.einsum('i,i->', probabilities, np.log2(probabilities))
```

이 코드는 ChatGPT 에게 물어봤을 때 알려준 코드이다. Einstein 합계를 사용하여 확률과 로그의 곱의 합계를 위와 같이 계산할 수 있다. Np.einsum = 확률 * Np.log2(확률)이라고 한다. 코드가 한결 간결해지고 깔끔해졌지만, einsum 함수를 모르는 사용자들에게는 가독성이 낮게 느껴지는 코드일 것 같다.

위 세 버전의 코드 모두 같은 식을 따르므로 결과는 전부 동일하게 나왔다.

2.3 Information Gain

2.3.1 기존 코드

Step 1: Calculate the entropy for the label in the dataset

$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} (P(t = l) \times log_2(P(t = l)))$$

Step 2: Calculate the remainder for each feature

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \underbrace{\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|}}_{\text{weighting}} \times \underbrace{H(t, \mathcal{D}_{d=l})}_{\text{entropy of partition } \mathcal{D}_{d=l}}$$

dataset

전체 데이터에서 l 라벨을 가진 d feature 비율

Step 3: Calculate the information gain for all features

$$IG(d, \mathcal{D}) = H(t, \mathcal{D}) - rem(d, \mathcal{D})$$

```
# 기존 코드
Weighted_Entropy = np.sum([(counts[i] / np.sum(counts)) *
entropy(data.where(data[split_attribute_name] ==
vals[i]).dropna()[target_name]) \
                           for i in range(len(vals))])
```

기존 코드는 위와 같이 weighted entropy 를 구하는 식이 구성되어 있었다. 이는 위 강의자료에서 제공된 식의 rem(d,D) 부분이다. H(t, D)는 2.2 의 코드로 구할 수 있다. 우선 np.sum 과 for 문으로 시그마가 구성되어 있는 것을 알 수 있다. 위와 다른 점은 식과 같이 vals 에 대해 for 문이 돌아간다는 것이다. 그 이후 전체 데이터에서 i 라벨을 가진 d feature 의 비율 부분을 counts[i] / np.sum(counts) 로 계산하고, entropy 함수에 각 d = l 일때의 attribute name 을 넣어 entropy 를 계산하였다. where 함수는 데이터셋을 필터링하여 코드 상의 vals[i]와 일치하는 행만 유지하도록 하는 함수이다. Dropna()는 누락된 값이 있는 행을 제거하는 함수라고 한다. 이렇게 rem(d,D) 부분의 식을 이러한 코드로 구성되어 있음을 분석하였다.

2.3.2 변경 코드 및 설명

```
# 변경 코드 1
Weighted_Entropy = np.sum([(count / np.sum(counts)) *
entropy(data[data[split_attribute_name] == val][:][target_name]) \
                           for val, count in zip(vals, counts)])
```

위 코드에서는 기존 vals 과 counts 의 반복을 인덱스 등을 사용하지 않고 조금 더 간결하게 하기 위해 구성한 코드이다. 전체적으로 동일한 함수를 사용한 코드이지만 for 문에 zip 을

사용하여 val 과 count 가 반복하도록 하였다. 그리고 where 함수와 dropna 함수를 사용하지 않고 slicing 으로 직접 target_name 을 필터링 하도록 코드를 작성하였다. n

```
# 변경 코드 2
Weighted_Entropy = np.sum((counts / np.sum(counts)) *
np.array([entropy(data[data[split_attribute_name] == val][target_name]) \
                                                    for val in vals]))
```

이 코드는 중간에 np.array를 사용하여서 배열을 생성하여 rem(d,D) 식을 계산하였다. Counts / np.sum(counts) 확률에 entropy함수와 for문으로 구성된 entropy 배열을 곱하고 np.sum으로 이를 합산하여 weigthed entropy를 계산하였다. Numpy는 python의 list와 달리 아주 빠른 언어인 C로 구성되어 있으므로 계산에 있어 훨씬 효율적이다. 지금보다 많은 데이터를 처리할 때에는 자원의 절약이 매우 중요하므로 이렇게 데이터를 계산하는 자료구조를 변경하여 시간을 절약하는 것도 매우 중요하다.

위 세 버전의 코드 모두 같은 식을 바탕으로 작성되어 있으므로 동일한 결과를 반환하였다.

3. 결론

3.1 고찰

본 과제에서는 ID3 algorithm을 기반으로 작성된 decision tree 코드에 대해서 분석하고 몇줄의 코드를 변경해보는 경험을 할 수 있었다. 다양한 numpy 함수들과 어려운 재귀 호출로 구성된 코드이니 만큼 해석하고 읽어보는 데에 많은 시간이 걸렸다.

```
{'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}}
예측 정확도는:  85.71428571428571 %
```

위는 기존 코드와 변경 코드 모두 돌렸을 때 출력된 동일한 결과의 decision tree와 예측 정확도이다. 코드와 수치를 계산하는 데에 쓰인 함수가 달라져도 결과는 전부 동일하게 나온 것을 볼 수 있다. 우선 전부 수업시간에 정의된 entropy의 식에 기반하여 작성된 것이므로 당연한 결과임을 유추할 수 있다. 특히 코드를 변경하면서, list나 dictionary를 numpy.array로 변경하는 것 만으로 수치 계산의 효율성이 높아질 수 있다는 사실을 알고, 이러한 것들을 숙지하는 것이 데이터 분석과 학습에 중요하다는 걸 깨달았다. 그리고 친절한 주석과 알고리즘이 모두 주어진 상태임에도 코드를 읽어보며 이해하는 것은 정말 어려운 과정임을 깨달았고 꾸준한 학습과 시도만이 성장할 수 있는 방법이라는 생각이 들었다. 본 과제를 통해 코드에 대한 이해도와 지식이 조금은 오른 것 같아, 더욱 더 공부해야겠다는 생각을 하였다.

4. 참고문헌

1) 인공지능개론 수업 자료
2) 교수님께서 제공해주신 코드와 데이터
3) 한권으로 다지는 머신러닝 & 딥러닝 with 파이썬 (알베르토 아르타산체스, 프라틱 조시 / 한빛미디어)
4) ChatGPT
Chat.openai.com
5) numpy documentation
https://numpy.org/doc/stable/index.html