Decision Trees: Manual Implementation of Gini & Entropy calculations to compute optimal splits, Custom Function

Calvin Tran

> Using the virtual machine and either R or Python, your choice,
>
> write a function to compute an optimal (it might not be unique) split point for the animal's sample dataset of 20 animals
>
> (This sample has 2 snakes, 3 birds, 2 gorillas, 3 dogs, 1 cow, 7 butterflies, and 2 caterpillars)
>
> Compare your initial Gini and entropy before and after the first split.

(My work begins here)

Goal: Compute a *single* split point that results in the lowest impurity and entropy possible

Strategy:

1. simulate the single split many times where each category will end up either on the right or the left
2. calculate the metrics for each simulated split
3. if the metrics are lowest, then save that split
4. compare best split for gini impurity and entropy to see if they are the same

then compare these results to the gini impurity and entropy for the data that is unsplit.

In [1]:
```
import numpy as np
import random
import math

sample = ['snake','snake','bird','bird','bird','gorilla','gorilla','dog','dog','dog','
          'butterfly','butterfly','butterfly','butterfly','butterfly','butterfly','but
          'catepillar','catepillar']

#view categories for example's sake
categories = np.unique(sample)
print(categories)
```

```
['bird' 'butterfly' 'catepillar' 'cow' 'dog' 'gorilla' 'snake']
```

Write function to compute optimal split

In [2]:
```
def splitter_funct(data):
    categories = np.unique(data)
    best_entropy = 10 #placeholder values since we want to check against lower values
    best_impurity = 10
    best_split_entropyleft = []
    best_split_entropyright = []
    best_split_impurityleft = []
    best_split_impurityright = []
    for i in range(0,10000): #Outer loop:create 10000 simulated splits to avoid need t
        left_assignment = [] #keep track of how categories will be split
        right_assignment = []
        left_split = 0       #count the individuals in each split who are sorted by ca
        right_split = 0
        for j in range(0,len(categories)): #Inner loop 1: loop over the category names
            random_split = random.randint(0,1) #randomly assign each category to right
            if random_split == 0:
                    left_assignment.append(categories[j])
            if random_split == 1:
                    right_assignment.append(categories[j])
        for k in range(0,len(data)): #Inner loop 2: loop over the data & check if assi
            if data[k] in left_assignment:
                left_split += 1      #increment split accordingly to calculate probabi
            if data[k] in right_assignment:
                right_split += 1
        #calculate probablities for left/right splits based on assignment for calculat
        p_left = left_split/len(data)
        p_right = right_split/len(data)
        #calculate entropy
        if p_left != 0 and p_right != 0: #cannot take the log of 0, we also need to ac
            entropy = ((p_left*-1)*(math.log(p_left,2))) + ((p_right*-1)*(math.log(p_r
            if entropy <= best_entropy: #check metric to see if it is the best and upd
                best_entropy = entropy
                best_split_entropyleft = left_assignment
                best_split_entropyright= right_assignment
        #calculate impurity
        if p_left != 0 and p_right != 0: #again, we need to actually have a split and
            impurity = (p_left * (1-p_left)) + (p_right *(1-p_right)) #calculate based
            if impurity <= best_impurity:
                best_impurity = impurity
                best_split_impurityleft = left_assignment
                best_split_impurityright = right_assignment
    print(f'The best gini impurity value is {best_impurity}.')
    print(f'The best entropy value is {best_entropy}.')
    print(f'the best splits according to impurity are {best_split_impurityleft} and {b
    print(f'the best splits according to entropy are {best_split_entropyleft} and {bes
    if best_split_entropyleft == best_split_impurityleft and best_split_impurityright
        print('These splits are the same!')
    else:
        print('These splits are different!')
    return best_impurity, best_entropy, best_split_impurityleft, best_split_impurityri
```

Use splitter function to compute optimal split

In [3]:
```
splitter_funct(sample)
```

```
The best gini impurity value is 0.09500000000000004.
The best entropy value is 0.28639695711595625.
the best splits according to impurity are ['cow'] and ['bird', 'butterfly', 'catepill
ar', 'dog', 'gorilla', 'snake'].
the best splits according to entropy are ['cow'] and ['bird', 'butterfly', 'catepilla
r', 'dog', 'gorilla', 'snake'].
These splits are the same!
```

Out[3]:
```
(0.09500000000000004,
 0.28639695711595625,
 ['cow'],
 ['bird', 'butterfly', 'catepillar', 'dog', 'gorilla', 'snake'],
 ['cow'],
 ['bird', 'butterfly', 'catepillar', 'dog', 'gorilla', 'snake'])
```

Calculate Gini Impurity and Entropy for non-split data

In [4]:
```
# calculate probabilities of sample
probability_list = []

for i in range(0,len(categories)):
    count = 0
    for j in range(0,len(sample)):
        if sample[j] == categories[i]:
            count +=1
    probability = count/len(sample)
    probability_list.append(probability)

print(probability_list)
```

```
[0.15, 0.35, 0.1, 0.05, 0.15, 0.1, 0.1]
```

In [5]:
```
#SAME THING IN ONLY 1 LINE OF CODE I AM A FOOL
probs_array = (np.unique(sample, return_counts=True)[1])/len(sample)
print(probs_array)
```

```
[0.15 0.35 0.1  0.05 0.15 0.1  0.1 ]
```

In [6]:
```
# Gini impurity
gini = 0
for i in range(0,len(probs_array)):
       gini += probs_array[i]*(1-probs_array[i])
print(gini)
```

```
0.7999999999999999
```

In [7]:
```
# Entropy
entropy = 0
for i in range(0,len(probs_array)):
    entropy += (probs_array[i]*-1)*(math.log(probs_array[i],2))
print(entropy)
```

```
2.5638651219508537
```

Comparison of split vs unsplit data

Unsplit Values:

- Gini Impurity: ~0.800
- Entropy: ~2.564

Split Values:

- Gini Impurity: ~0.095
- Entropy: ~0.286

These values went down quite a lot as a result of the split, which is our goal. The result is the split values are about one-tenth the unsplit values.