index

March 20, 2022

1 ID3 Classification Trees: Perfect Split with Information Gain - Lab

1.1 Introduction

In this lab, we will simulate the example from the previous lesson in Python. You will write functions to calculate entropy and IG which will be used for calculating these uncertainty measures and deciding upon creating a split using information gain while growing an ID3 classification tree. You will also write a general function that can be used for other (larger) problems as well. So let's get on with it.

1.2 Objectives

In this lab you will:

- Write functions for calculating entropy and information gain measures
- Use entropy and information gain to identify the attribute that results in the best split at each node

1.3 Problem

You will use the same problem about deciding whether to go and play tennis on a given day, given the weather conditions. Here is the data from the previous lesson:

outlook	temp	humidity	windy	play
overcast	cool	high	Y	yes
overcast	mild	normal	N	yes
sunny	cool	normal	N	yes
overcast	hot	high	Y	no
sunny	hot	normal	Y	yes
rain	mild	high	N	no
rain	cool	normal	N	no
sunny	mild	high	N	yes
sunny	cool	normal	Y	yes
sunny	mild	normal	Y	yes
overcast	cool	high	N	yes
rain	cool	high	Y	no
sunny	hot	normal	Y	no

outlook	temp	humidity	windy	play
sunny	mild	high	N	yes

1.4 Write a function `entropy(pi)` to calculate total entropy in a given discrete probability distribution `pi`

- The function should take in a probability distribution `pi` as a list of class distributions. This should be a list of two integers, representing how many items are in each class. For example: `[4, 4]` indicates that there are four items in each class, `[10, 0]` indicates that there are 10 items in one class and 0 in the other.
- Calculate and return entropy according to the formula:

$$Entropy(p) = -\sum(P_i.log_2(P_i))$$

[11]:
```
from math import log

def entropy(pi):
    """
    return the Entropy of a probability distribution:
    entropy(p) = - SUM (Pi * log(Pi) )
    """

    total = 0

    for p in pi:

        m = p / sum(pi)

        if m != 0:

            total += -1 * m*log(m,2)

        else:

            total += 0

    return total


# Test the function

print(entropy([1, 1])) # Maximum Entropy e.g. a coin toss
print(entropy([0, 6])) # No entropy, ignore the -ve with zero , it's there due␣
 ↪to log function
print(entropy([2, 10])) # A random mix of classes

# 1.0
```

```
# -0.0
# 0.6500224216483541
```

```
1.0
0.0
0.6500224216483541
```

1.5 Write a function IG(D,a) to calculate the information gain

- As input, the function should take in `D` as a class distribution array for target class, and `a` the class distribution of the attribute to be tested
- Using the `entropy()` function from above, calculate the information gain as:

$$gain(D, A) = Entropy(D) - \sum(\frac{|D_i|}{|D|}.Entropy(D_i))$$

where D_i represents distribution of each class in `a`.

[16]:
```
def IG(D, a):
    """
    return the information gain:
    gain(D, A) = entropy(D)- SUM( |Di| / |D| * entropy(Di) )
    """
    ent_D = entropy(D)

    total = 0

    for i in a:

        total += np.abs(sum(i)/sum(D)) * entropy(i)

    ig = ent_D - total
    return ig


# Test the function
# Set of example of the dataset - distribution of classes
test_dist = [6, 6] # Yes, No
# Attribute, number of members (feature)
test_attr = [ [4,0], [2,4], [0,2] ] # class1, class2, class3 of attr1 according␣
 ↪to YES/NO classes in test_dist

print(IG(test_dist, test_attr))

# 0.5408520829727552
```

```
0.5408520829727552
```

1.6 First iteration - Decide the best split for the root node

- Create the class distribution `play` as a list showing frequencies of both classes from the dataset
- Similarly, create variables for four categorical feature attributes showing the class distribution for each class with respect to the target classes (yes and no)
- Pass the play distribution with each attribute to calculate the information gain

[17]:
```
# Your code here


play = [9, 5]

outlook = [
    [3, 1],  # overcast   [yes, no]
    [6, 1],  # sunny
    [0, 3]   # rain
]

temperature = [
    [4, 2],   # cool    [yes, no]
    [4, 1],   # mild    [yes, no]
    [1, 2]    # hot     [yes, no]
]

humidity = [
    [4, 3],   # high      [yes, no]
    [5, 2]    # normal    [yes, no]
]

wind = [
    [4, 3],   # Y      [yes, no]
    [5, 2]    # N    [yes, no]
]
# Information Gain:

print ('Information Gain:\n' )
print('Outlook:', IG(play, outlook))
print('Temperature:', IG(play, temperature))
print('Humidity:', IG(play, humidity))
print('Wind:,', IG(play, wind))

# Outlook: 0.41265581953400066
# Temperature: 0.09212146003297261
# Humidity: 0.0161116063701896
# Wind:, 0.0161116063701896
```

```
Information Gain:
```

```
Outlook: 0.41265581953400066
Temperature: 0.0921214600329725
Humidity: 0.0161116063701896
Wind:, 0.0161116063701896
```

We see here that the outlook attribute gives us the highest value for information gain, hence we choose this for creating a split at the root node. So far, we've built the following decision tree:

1.7 Second iteration

Since the first iteration determines what split we should make for the root node of our tree, it's pretty simple. Now, we move down to the second level and start finding the optimal split for each of the nodes on this level. The first branch (edge) of three above that leads to the "Sunny" outcome. Of the temperature, humidity and wind attributes, find which one provides the highest information gain.

Follow the same steps as above. Remember, we have 6 positive examples and 1 negative example in the "sunny" branch.

[18]:
```
# Your code here
play = [6, 1]  # sunny [yes, no]


temperature = [
    [2, 0],   # cool    [yes, no]
    [3, 0],   # mild    [yes, no]
    [1, 1]    # hot     [yes, no]
]

humidity = [
    [2, 0],   # high      [yes, no]
    [4, 1]    # normal    [yes, no]
]

wind = [
    [3, 1],   # Y      [yes, no]
    [3, 0]    # N    [yes, no]
]



# Information Gain:
print ('Information Gain:\n' )

print('Temperature:', IG(play, temperature))
print('Humidity:', IG(play, humidity))
print('Wind:,', IG(play, wind))

# Temperature: 0.3059584928680418
```

```
# Humidity: 0.07600985366627829
# Wind: 0.12808527889139443
```

```
Information Gain:

Temperature: 0.3059584928680418
Humidity: 0.07600985366627829
Wind:, 0.12808527889139443
```

We see that temperature gives us the highest information gain, so we'll use it to split our tree as shown below:

Let's continue.

1.8 Third iteration

We'll now calculate splits for the 'temperature' node we just created for days where the weather is sunny. Temperature has three possible values: [Hot, Mild, Cool]. This means that for each of the possible temperatures, we'll need to calculate if splitting on windy or humidity gives us the greatest possible information gain.

Why are we doing this next instead of the rest of the splits on level 2? Because a decision tree is a greedy algorithm, meaning that the next choice is always the one that will give it the greatest information gain. In this case, evaluating the temperature on sunny days gives us the most information gain, so that's where we'll go next.

1.9 All other iterations

What happens once we get down to a 'pure' split? Obviously, we stop splitting. Once that happens, we go back to the highest remaining uncalculated node and calculate the best possible split for that one. We then continue on with that branch, until we have exhausted all possible splits or we run into a split that gives us 'pure' leaves where all 'play=Yes' is on one side of the split, and all 'play=No' is on the other.

1.10 Summary

This lab should have helped you familiarize yourself with how decision trees work 'under the hood', and demystified how the algorithm actually 'learns' from data by:

- Calculating entropy and information gain
- Figuring out the next split you should calculate ('greedy' approach)