WEEK-11

October 25, 2023

[3]:
```
import numpy as np
import pandas as pd
from numpy import log2 as log
eps = np.finfo(float).eps
```

eps' here is the smallest representable number. At times we get log(0) or 0 in the denominator, to avoid that we are going to use this.

USING ID3

[37]:
```
df = pd.read_csv('weather.csv')
df=df.drop(columns=['Day'])
print(df)
```

```
     Outlook  Temp  Humidity    Wind Decision
0      Sunny    85        85    Weak       No
1      Sunny    80        90  Strong       No
2   Overcast    83        78    Weak      Yes
3       Rain    70        96    Weak      Yes
4       Rain    68        80    Weak      Yes
5       Rain    65        70  Strong       No
6   Overcast    64        65  Strong      Yes
7      Sunny    72        95    Weak       No
8      Sunny    69        70    Weak      Yes
9       Rain    75        80    Weak      Yes
10     Sunny    75        70  Strong      Yes
11  Overcast    72        90  Strong      Yes
12  Overcast    81        75    Weak      Yes
13      Rain    71        80  Strong       No
```

1. claculate entropy o the whole dataset

[38]:
```
def find_entropy(df):
    Class = df.keys()[-1]    #To make the code generic, changing target variable␣
 ↪class name
    entropy_node = 0   #Initialize Entropy
    values = df[Class].unique()   #Unique objects - 'Yes', 'No'
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df.Decision)
```

```
        entropy_node += -fraction*np.log2(fraction)
    print(f"Entropy_Node-> {entropy_node}")
    return(entropy_node)
```

2 .Now define a function {ent} to calculate entropy of each attribute :

[39]:
```
def ent(df,attribute):
    target_variables = df.Decision.unique()  #This gives all 'Yes' and 'No'
    variables = df[attribute].unique()    #This gives different features in␣
 ↪that attribute (like 'Sweet')


    entropy_attribute = 0
    for variable in variables:
        entropy_each_feature = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df.Decision␣
 ↪==target_variable]) #numerator
            den = len(df[attribute][df[attribute]==variable])  #denominator
            fraction = num/(den+eps)  #pi
            entropy_each_feature += -fraction*log(fraction+eps) #This␣
 ↪calculates entropy for one feature like 'Sweet'
        fraction2 = den/len(df)
        entropy_attribute += -fraction2*entropy_each_feature   #Sums up all the␣
 ↪entropy ETaste

    return(abs(entropy_attribute))
```

find_entropy_attribute

[40]:
```
def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]   #To make the code generic, changing target variable␣
 ↪class name
  target_variables = df[Class].unique()  #This gives all 'Yes' and 'No'
  variables = df[attribute].unique()    #This gives different features in that␣
 ↪attribute (like 'Hot','Cold' in Temperature)
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][df[Class]␣
 ↪==target_variable])
          den = len(df[attribute][df[attribute]==variable])
          fraction = num/(den+eps)
          entropy += -fraction*log(fraction+eps)
      fraction2 = den/len(df)
      entropy2 += -fraction2*entropy
```

```
  return abs(entropy2)
```

find_winner

[41]:
```
def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:#Entropy_att.
 ↪append(find_entropy_attribute(df,key))
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]
```

get_subtable

[42]:
```
def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)
```

Build a Tree

[43]:
```
def buildTree(df,tree=None):
    Class = df.keys()[-1]   #To make the code generic, changing target variable␣
 ↪class name

    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)

    #Get distinct value of that attribute e.g Salary is node and Low,Med and␣
 ↪High are values
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

   #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable[Class],return_counts=True)

        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
```

```
            tree[node][value] = buildTree(subtable) #Calling the function␣
↪recursively

    return tree
```

Final Output

[44]:
```
import pprint
T= buildTree(df)
pprint.pprint(T)
```

```
Entropy_Node-> 0.9402859586706311
Entropy_Node-> 0.9402859586706311
Entropy_Node-> 0.9402859586706311
Entropy_Node-> 0.9402859586706311
Entropy_Node-> 1.0
Entropy_Node-> 1.0
Entropy_Node-> 1.0
Entropy_Node-> 1.0
{'Temp': {64: 'Yes',
          65: 'No',
          68: 'Yes',
          69: 'Yes',
          70: 'Yes',
          71: 'No',
          72: {'Outlook': {'Overcast': 'Yes', 'Sunny': 'No'}},
          75: 'Yes',
          80: 'No',
          81: 'Yes',
          83: 'Yes',
          85: 'No'}}
```

USING CART ALGORITHM

[51]:
```
df = pd.read_csv('weather.csv')
df=df.drop(columns=['Day'])
df
```

[51]:

	Outlook	Temp	Humidity	Wind	Decision
0	Sunny	85	85	Weak	No
1	Sunny	80	90	Strong	No
2	Overcast	83	78	Weak	Yes
3	Rain	70	96	Weak	Yes
4	Rain	68	80	Weak	Yes
5	Rain	65	70	Strong	No
6	Overcast	64	65	Strong	Yes
7	Sunny	72	95	Weak	No

```
8      Sunny    69         70    Weak      Yes
9       Rain    75         80    Weak      Yes
10     Sunny    75         70  Strong      Yes
11  Overcast    72         90  Strong      Yes
12  Overcast    81         75    Weak      Yes
13      Rain    71         80  Strong       No
```

[52]:
```
data=pd.read_csv('weather.csv')
data=data.drop(columns=['Day'])
attributes = data.columns[1:-1]
target_attribute = 'Decision'
class Node:
    def __init__(self, attribute=None, value=None, result=None):
        self.attribute = attribute
        self.value = value
        self.result = result
        self.children = {}
def gini_impurity(data, target_attribute):
    value_counts = data[target_attribute].value_counts()
    total = len(data)
    impurity = 1
    for value_count in value_counts:
        probability = value_count / total
        impurity -= probability ** 2
    return impurity

def gini_impurity_gain(data, attribute, target_attribute):
    impurity_before = gini_impurity(data, target_attribute)
    values, counts = np.unique(data[attribute], return_counts=True)
    impurity_after = 0
    for value, count in zip(values, counts):
        subset = data[data[attribute] == value]
        impurity_after += (count / len(data)) * gini_impurity(subset,␣
↪target_attribute)
    return impurity_before - impurity_after

def cart(data, attributes, target_attribute):
    if len(np.unique(data[target_attribute])) == 1:
        return Node(result=data[target_attribute].iloc[0])

    if len(attributes) == 0:
        most_common = data[target_attribute].value_counts().idxmax()
        return Node(result=most_common)

    best_attribute = max(attributes, key=lambda a: gini_impurity_gain(data, a,␣
↪target_attribute))
    tree = Node(attribute=best_attribute)
```

```
    values = np.unique(data[best_attribute])
    for value in values:
        subset = data[data[best_attribute] == value]
        if len(subset) == 0:
            most_common = data[target_attribute].value_counts().idxmax()
            tree.children[value] = Node(result=most_common)
        else:
            remaining_attributes = [a for a in attributes if a !=␣
↪best_attribute]
            tree.children[value] = cart(subset, remaining_attributes,␣
↪target_attribute)

    return tree

tree = cart(data, attributes, target_attribute)

new_sample = pd.Series({'Outlook': 'Sunny', 'Temp': 69, 'Humidity': 80, 'Wind':␣
↪'Strong'})

def classify(tree, sample):
    if tree.result is not None:
        return tree.result
    attribute_value = sample[tree.attribute]
    if attribute_value in tree.children:
        return classify(tree.children[attribute_value], sample)

result = classify(tree, new_sample)
print("Predicted decision:", result)
```

```
Predicted decision: Yes
```

USING C4.5

[56]:

```
def information_gain(data, attribute, target_attribute):
    total_entropy =  find_entropy_attribute(data, target_attribute)
    values, counts = np.unique(data[attribute], return_counts=True)
    weighted_entropy = 0
    for value, count in zip(values, counts):
        subset = data[data[attribute] == value]
        weighted_entropy += (count / len(data)) * ␣
↪find_entropy_attribute(subset, target_attribute)
    return total_entropy - weighted_entropy

def c45(data, attributes, target_attribute, parent_data, parent_value=None):
    if len(np.unique(data[target_attribute])) == 1:
        return Node(result=data[target_attribute].iloc[0])
```

```python
    if len(attributes) == 0:
        most_common = data[target_attribute].value_counts().idxmax()
        return Node(result=most_common)

    best_attribute = max(attributes, key=lambda a: information_gain(data, a, target_attribute))
    tree = Node(attribute=best_attribute)

    values = np.unique(data[best_attribute])
    for value in values:
        subset = data[data[best_attribute] == value]
        if len(subset) == 0:
            most_common = data[target_attribute].value_counts().idxmax()
            tree.children[value] = Node(result=most_common)
        else:
            remaining_attributes = [a for a in attributes if a != best_attribute]
            tree.children[value] = c45(subset, remaining_attributes, target_attribute, data, value)

    return tree

data = pd.read_csv('weather.csv')

attributes = data.columns[1:-1]
target_attribute = 'Decision'

tree = c45(data, attributes, target_attribute, None)

new_sample = pd.Series({'Outlook': 'Sunny', 'Temp': 85, 'Humidity': 85, 'Wind': 'Weak'})

def classify(tree, sample, parent_value=None):
    if tree.result is not None:
        return tree.result
    attribute_value = sample[tree.attribute]
    if attribute_value in tree.children:
        return classify(tree.children[attribute_value], sample, attribute_value)
    else:
        # If the attribute value is not found, use the parent's value if available
        if parent_value is not None:
            return classify(tree.children[parent_value], sample, parent_value)
        else:
            most_common = data[target_attribute].value_counts().idxmax()
            return most_common
```

```python
result = classify(tree, new_sample)
print("Predicted decision:", result)
```

```
Predicted decision: No
```

[ ]:

```python

```