27/2/23

EXPERIMENT-4

AIM

To implement the C4.5 algorithm in Python

ALGORITHM

* Compute Entropy_Info for the whole training dataset based on the target attribute
* Compute Entropy_Info, Info_Gain, Split_Info and Gain_Ratio for each of the attribute in the training dataset
* Choose the best split attribute for which Gain_Ratio is maximum and place as root node
* Each subtree is an outcome of the test condition of root node attribute. Accordingly, training dataset is also split into subsets
* Recursively apply with the remaining attributes until a leaf node is derived or no more training instances are available in the subset

THEORY

C4.5 is based on Occam's Razor which says that given 2 correct solutions, the simpler has to be chosen. It uses Gain_Ratio as a measure during construction of decision trees.

$$\text{Gain\_Ratio}(A) = \frac{\text{Info\_Gain}(A)}{\text{Split\_Info}(T, A)}$$

OUTPUT

```
Data read from marks.csv :

   CGPA Interactiveness Practical Knowledge Communication Skill Job Offer
0     9             Yes           Very good                Good       Yes
1     8              No                Good            Moderate       Yes
2     9              No             Average                Poor        No
3     7              No             Average                Good        No
4     8             Yes                Good            Moderate       Yes
5     9             Yes                Good            Moderate       Yes
6     7             Yes                Good                Poor        No
7     9              No           Very good                Good       Yes
8     8             Yes                Good                Good       Yes
9     8             Yes             Average                Good       Yes

 CGPA                  Gain = 0.5567796494470396      Split = 1.5219280948873621
Interactiveness        Gain = 0.09127744624168022     Split = 0.9709505944546686
Practical Knowledge    Gain = 0.24483810157066466     Split = 1.4854752972273344
Communication Skill    Gain = 0.5203268517870115      Split = 1.4854752972273344

 CGPA                  Gain = 0.0                     Split = 0.0
Interactiveness        Gain = 0.31127812445913283     Split = 1.0
Practical Knowledge    Gain = 0.8112781244591328      Split = 1.5
Communication Skill    Gain = 0.8112781244591328      Split = 1.5
```

```
 Final TREE
 ----------

CGPA
    9
        Practical Knowledge
            Very good
                Yes
            Average
                No
            Good
                Yes
    8
        Yes
    7
        No
```

CODE

```
import pandas as pd
import numpy as np

def calc_total_entropy(data,target_name,target_val):
    size = data.shape[0]
    total_entropy=0
    for i in target_val:
        count = data[data[target_name] == i].shape[0]
        entropy = -(count/size)*np.log2(count/size)
        total_entropy += entropy
    return total_entropy

def calc_entropy(data,target_name,target_val):
    size = data.shape[0]
    total_entropy=0
    for i in target_val:
        count = data[data[target_name] == i].shape[0]
        entropy = 0
        if count !=0:
            entropy = -(count/size)*np.log2(count/size)
        total_entropy += entropy
    return total_entropy

def calc_split_info(split):
    split_info = 0
    for el in split :
        split_info += -(el*np.log2(el))
    return split_info

def calc_gain(attribute,data,target_name,target_val):
    list_ = data[attribute].unique()
    size = data.shape[0]
    gain = 0.0
    probs = []
    for i in list_:
        t_data = data[data[attribute] == i]
        count = data[data[attribute] == i].shape[0]
        entropy = calc_entropy(t_data, target_name, target_val)
        prob = count/size
        probs.append ( prob )
        gain += prob*entropy
    split = calc_split_info(probs)
    return calc_total_entropy(data, target_name, target_val) - gain, split

def ratio_attribute(data,target_name,target_val):
    list_ = data.columns.drop(target_name)
    max_ratio = -1
    info_feat = None
    matrix = []
    for i in list_:
        gain, split = calc_gain(i,data,target_name,target_val)
```

```
        matrix.append([i, "Gain = " + str(gain), "Split = " + str(split)])
        ratio = gain / split if split > 0 else 0
        if max_ratio < ratio:
            max_ratio = ratio
            info_feat = i
    s = [[str(e) for e in row] for row in matrix]
    lens = [max(map(len, col)) for col in zip(*s)]
    fmt = '\t'.join('{{:{}}}'.format(x) for x in lens)
    table = [fmt.format(*row) for row in s]
    print ( "\n".join(table) , "\n" )
    return info_feat

def generate_tree(attribute,data,target_name,target_val):
    count_dict=data[attribute].value_counts(sort = False)
    tree = {}
    for value,count in count_dict.items():
        feat_data = data[data[attribute] == value]
        pure = False
        for t in target_val:
            class_count = feat_data[feat_data[target_name] == t].shape[0]
            if class_count == count:
                tree[value] = t
                data = data[data[attribute] != value]
                pure = True
        if not pure:
            tree[value] = '?'
    return tree,data

def make_tree(root,prev,data,target_name,target_val):
    if data.shape[0] != 0:
        max_info_attribute = ratio_attribute(data, target_name, target_val)
        tree,data = generate_tree(max_info_attribute, data, target_name, target_val)
        next_root = None
        if prev != None:
            root[prev] = dict()
            root[prev][max_info_attribute] = tree
            next_root = root[prev][max_info_attribute]
        else:
            root[max_info_attribute] = tree
            next_root = root[max_info_attribute]
        for node,branch in list(next_root.items()):
            if branch == "?":
                feat_data = data[data[max_info_attribute] == node]
                make_tree ( next_root,node,feat_data,target_name,target_val )

def c4_5(data,target_name):
    tree = {}
    target = data[target_name].unique()
    make_tree(tree,None,data,target_name,target)
    return tree

def print_tree ( node , indent = 0 ) :
    for key , value in node.items () :
```

```
        print ( ' ' * indent + str ( key ) )
        if isinstance ( value , dict ) :
            print_tree ( value , indent + 5 )
        else :
            print ( ' ' * ( indent + 5 ) + str ( value ) )

data = pd.read_csv ( 'marks.csv' )
print ( "\n Data read from marks.csv :\n\n" , data )
tree = c4_5 ( data , 'Job Offer' )
print ( "\n Final TREE\n ----------\n" )
print_tree ( tree )
```

RESULT

Hence, C4.5 algorithm has been implemented successfully in Python