12/10/20

EXPERIMENT NO. 3

1KS17CS030

AIM: Write a program to demonstrate the working of decision tree based ID3 algorithm. Use an appropriate dataset for building the decision tree & apply this knowledge to classify a new sample

Dataset:

Outlook	Temparature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong.	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal.	Weak	Yes
Rain	Cool	Normal.	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain.	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong.	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild.	High.	Strong	No

Algorithm:

ID3 (Algorithm, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given examples.

- Create a root node for the tree
- If all Examples are positive, Return the single-node-tree Root, with label = +ve
- If all Examples are negative, Return the single-node tree Root, with label = -ve
- If attribute is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples
  - Otherwise Begin.

```
o A ← the attribute from Attributes the best* classifies Examples.
o The decision attribute for Root ← A.
o For each possible value, Vi of A.
    → Add a new tree branch below Root, corresponding to the test
              A = Vi
    → Let Examples Vi be the subset of Examples that have value Vi for A.
    → If Examples Vi is empty.
        • Then below this new branch add a leaf node with label =
          most common value of Target_attributes in Examples.
        • Else below this new branch add the subtree
                ID3 (Examples Vi, Target_address, Attributes − {A})
  END.
Return Root
```

PROGRAM:

```
import math
import csv

def majorclass (attributes, data, target):
    freq = { }
    index = attributes.index(target)
    for tuple in data:
        if tuple[index] in freq:
            freq [tuple[index]] += 1.
        else:
            freq [tuple[index]] = 1.
    max = 0
    major = " "
    for key in freq.keys():
        if freq[key] > max:
            max = freq[key]
            major = key.
    return major

def entropy(attributes, data, targetAttr):
    freq = { }.
    dataEntropy = 0.0
    i=0
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]] = 1.0.
    for freq in freq.values():
        dataEntropy += (-freq/len(data)) * math.log(freq/len(data)),2)
    return dataEntropy.

def info_gain(attributes, data, attr, targetAttr):
    freq = { }
    subsetEntropy = 0.0
    i = attributes.index(attr)
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]] = 1.0
    for val in freq.keys():
        valProb = freq[val] / sum(freq.values())
        dataSubset = [entry for entry in data if entry[i] == val]
```

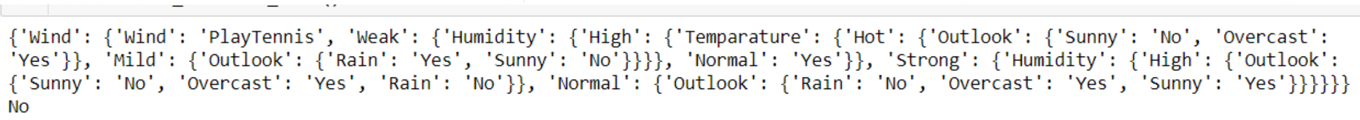
```
            subsetEntropy += valProb * entropy(attributes, dataSubset, targetAttr)
        return (entropy(attributes, data, targetAttr) - subsetEntropy)
def attr_choose(data, attributes, target):
        best = attribute[0]
        max Gain = 0
        for attr in attributes:
                newGain = info_gain(attributes, data, attr, target)
                if new Gain > max Gain:
                        max Gain = new Gain
                        best = attr.
        return best
def get_values(data, attributes, attr):
        index = attributes.index(attr)
        values = [ ]
        for entry in data:
                if entry[index] not in values:
                        values.append(entry[index])
        return values.
def get_data(data, attributes, best, val):
        new_data = [[]]
        index = attributes.index(best).
        for entry in data:
                if entry[index] == val:
                        newEntry = [ ]
                        for i in range(0, len(entry)):
                                if (i != index):
                                        newEntry.append(entry[i])
                        new_data.append(newEntry).
        new_data.remove ( [] )
        return new_data

def   build_tree (data, attributes, target ):
        data = data[:]
        vals = [record[attributes.index(target)] for record in data]
        default = majorclass (attributes, data, target)
        if  not  data  or  len(attributes) -1 <=0:
                return  default
        elif  vals.count (vals[0]) == len(vals):
                return  vals[0]
        else:
            best = attr_choose (data, attributes, target)
            tree = {best:{}}
            for  val  in  get_values (data, attributes, best):
                new_data = get_data (data, attributes, best, val)
                newAttr = attributes [:]
                newAttr.remove (best)
                subtree = build_tree (new_data, newAttr, target)
                tree [best][val] = subtree
        return  tree.

def   execute_decision_tree ():
      data = []
      with open ("weather.csv") as tsv:
            for  line  in  csv.reader (tsv):
                        data.append (tuple(line))
      attributes = ['outlook', 'Temparature', 'Humidity', 'Wind', 'PlayTennis']
      target = attributes [-1]
      acc = []
      training_set = [x for i, x in enumerate (data)]
      tree = build_tree (training_set, attributes, target)
```

```
print(tree)
results =[]
test_set= [('Sunny', 'Hot', 'High', 'Weak')]
for entry in test_set:
    tempDict = tree.copy()
    result =" '
    while (isinstance(tempDict, dict)):
        child = []
        nodeval= next(iter(tempDict))
        child = tempDict[next(iter(tempDict))].keys()
        tempDict = tempDict[next(iter(tempDict))]
        index= attributes.index(nodeval)
        value= entry[index]
        if value in tempDict.keys():
            result= tempDict[value]
            tempDict = tempDict[value]
        else:
            result = "Null"
            break
    if result != " Null"
        results.append(result == entry[-1])
        print(result)
if __name__ == "__main__":
    execute_decision_tree()
```

Output:



```
{'Wind': {'Wind': 'PlayTennis', 'Weak': {'Humidity': {'High': {'Temparature': {'Hot': {'Outlook': {'Sunny': 'No', 'Overcast':
'Yes'}}, 'Mild': {'Outlook': {'Rain': 'Yes', 'Sunny': 'No'}}}}, 'Normal': 'Yes'}}, 'Strong': {'Humidity': {'High': {'Outlook':
{'Sunny': 'No', 'Overcast': 'Yes', 'Rain': 'No'}}, 'Normal': {'Outlook': {'Rain': 'No', 'Overcast': 'Yes', 'Sunny': 'Yes'}}}}}}
No
```