-

1. EXPERIMENT NO: 3

2. TITLE: **ID3 ALGORITHM**

3. LEARNING OBJECTIVES:
   - Make use of Data sets in implementing the machine learning algorithms.
   - Implement ML concepts and algorithms in Python

4. AIM:
   - Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.

5. THEORY:
   - ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?".
   - To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.
   - A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute).
   - The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
   - A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described below.

Algorithm: ID3(Examples, TargetAttribute, Attributes)
Input: Examples are the training examples.
Targetattribute 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.
Output: Returns a decision tree that correctly classiJies the given Examples
Method:
1. Create a Root node for the tree
2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,
   Return the single-node tree Root, with label = most common value of TargetAttribute in Examples
   Else
   A ← the attribute from Attributes that best classifies Examples
   The decision attribute for Root ←A
   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 TargetAttribute in Examples
   Else
   below this new branch add the subtree ID3($Examples_{vi}$, TargetAttribute, Attributes–{A})
   End
5. Return Root

6. PROCEDURE / PROGRAMME :

```
import math
import csv

def load_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""     # NULL indicates children exists.
                             # Not Null indicates this is a Leaf Node

def subtables(data, col, delete):
    dic = {}
    coldata = [ row[col] for row in data]
    attr = list(set(coldata)) # All values of attribute retrived

    #counts = [0]*len(attr)
    #r = len(data)
    #c = len(data[0])
    #for x in range(len(attr)):
    #   for y in range(r):
    #       if data[y][col] == attr[x]:
    #           counts[x] += 1

    #for x in range(len(attr)):
      #dic[attr[x]] =  [[0 for i in range(c)] for j in range(counts[x])]
      #pos = 0
       #for y in range(r):
       #    if data[y][col] == attr[x]:
       #        if delete:
       #            del data[y][col]
       #        dic[attr[x]].append(data[y])
       #        #pos += 1

    for k in attr:
        dic[k] = []

    for y in range(len(data)):
        key = data[y][col]
        if delete:
            del data[y][col]
        dic[key].append(data[y])

    return attr, dic

def entropy(S):
    attr = list(set(S))
    if len(attr) == 1: #if all are +ve/-ve then entropy = 0
        return 0

    counts = [0,0] # Only two values possible 'yes' or 'no'
```

```
    for i in range(2):
        counts[i] = sum( [1 for x in S if attr[i] == x] ) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)

    return sums

def compute_gain(data, col):
    attValues, dic = subtables(data, col, delete=False)

    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attValues)):
        ratio = len(dic[attValues[x]]) / ( len(data) * 1.0)
        entro = entropy([row[-1] for row in dic[attValues[x]]])
        total_entropy -= ratio*entro

    return total_entropy

def build_tree(data, features):

    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1:   # If all samples have same labels return that label
        node=Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0])-1
    #gains = [0]*n
    #for col in range(n):
    #    gains[col] = compute_gain(data, col)
    gains = [compute_gain(data, col) for col in range(n) ]

    split = gains.index(max(gains)) # Find max gains and returns index
    node = Node(features[split])  # 'node' stores attribute selected
    #del (features[split])
    fea = features[:split]+features[split+1:]

    attr, dic = subtables(data, split, delete=True) # Data will be spilt in subtables
    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))

    return node

def print_tree(node, level):
    if node.answer != "":
        print("   "*level, node.answer) # Displays leaf node yes/no
        return

    print("   "*level, node.attribute)  # Displays attribute Name
    for value, n in node.children:
        print("   "*(level+1), value)
        print_tree(n, level + 2)

def classify(node,x_test,features):
    if node.answer != "":
        print(node.answer)
```

```
        return

    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

''' Main program '''
dataset, features = load_csv("data3.csv") # Read Tennis data
node = build_tree(dataset, features)    # Build decision tree

print("The decision tree for the dataset using ID3 algorithm is ")
print_tree(node, 0)

testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance :  ",xtest)
    print("The predicted label :  ", end="")
    classify(node,xtest,features)
```

7. RESULTS & CONCLUSIONS:

Training instances: data3.csv

```
Outlook,Temperature,Humidity,Wind,Target
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
```

Testing instances: data3_test.csv

```
Outlook,Temperature,Humidity,Wind
rain,cool,normal,strong
sunny,mild,normal,strong
```

Output

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
     overcast
         yes
     rain
         Wind
             weak
                 yes
             strong
                 no
     sunny
         Humidity
             normal
```

```
            yes
          high
            no
The test instance    :  ['rain', 'cool', 'normal', 'strong']
The predicted label :  no
The test instance    :  ['sunny', 'mild', 'normal', 'strong']
The predicted label :  yes
```

8. LEARNING OUTCOMES :

- The student will be able to demonstrate the working of the decision tree based ID3 algorithm, use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.

9. APPLICATION AREAS:

- Classification related prblem areas

10. REMARKS: