Top of Form

Bottom of Form

import math

class Node :

def \_\_init\_\_(self,pname=None,data=None,next\_node=[]) :

self.pname=pname

self.data=data

self.next\_node=next\_node

def printf(self):

print (self.pname)

print (self.data)

for i in range(len(self.next\_node)):

print (self.next\_node[i])

def setNextNode(self,val):

self.next\_node=val

HEAD=Node(None,None,None)

head=HEAD

listcat=['age','competition','type','profit','abc']

# datalist=[['Sunny','Hot','High','Weak',0],

# ['Sunny','Hot','High','Strong',0],

# ['Overcast','Hot','High','Weak',1],

# ['Rain','Mild','High','Weak',1],

# ['Rain','Cool','Normal','Weak',1],

# ['Rain','Cool','Normal','Strong',0],

# ['Overcast','Cool','Normal','Strong',1],

# ['Sunny','Mild','High','Weak',0],

# ['Sunny','Cool','Normal','Weak',1],

# ['Rain','Mild','Normal','Weak',1],

# ['Sunny','Mild','Normal','Strong',1],

# ['Overcast','Mild','High','Strong',1],

# ['Overcast','Hot','Normal','Weak',1],

# ['Rain','Mild','High','Strong',0]]

datalist=[['old', 'yes' ,'software',0],

['old', 'no' ,'software',0],

['old', 'no' ,'hardware',0],

['mid', 'yes' ,'software',0],

['mid', 'yes' ,'hardware',0],

['mid', 'no' ,'hardware',1],

['mid', 'no' ,'software',1],

['new', 'yes' ,'software',1],

['new', 'no' ,'hardware',1],

['new', 'no' ,'software',1]]

def predict(array,start):

for i in range(0,len(array)):

for j in range(len(start.next\_node)):

if start.next\_node[j].data==array[i]:

start=start.next\_node[j]

for k in range(len(start.next\_node)):

if start.next\_node[k].data=='0':

return 0

elif start.next\_node[k].data=='1':

return 1

break

def printTree(start):

if start.next\_node==None:

return

else:

for j in range(len(start.next\_node)):

print(start.next\_node[j].data)

printTree(start.next\_node[j])

def maximum(array):

# print(array)

if len(array)>1:

tdata=array[0][1]

pos=array[0][0]

min=array[0][2]

for i in range(1,len(array)):

#print(array[i][2])

if array[i][2]>min:

min=array[i][2]

tdata=array[i][1]

pos=array[i][0]

# print [pos,tdata,min]

return [pos,tdata,min]

else:

return array[0]

#adds string to tree

def addNode(start,string,list1,colNo):

#starting of tree

if start.next\_node==None and start==head:

start.data=string

node\_list=[]

for i in range(len(getSubArgs(list1,colNo))):

node\_list.append(Node(string,getSubArgs(list1,colNo)[i],None))

start.setNextNode(node\_list)

#if the prenet node is matched

elif start.data==string:

node\_list=[]

for i in range(len(getSubArgs(list1,colNo))):

node\_list.append(Node(string,str(getSubArgs(list1,colNo)[i]),None))

start.setNextNode(node\_list)

#if we reach the root node but Still couldnot match the string

elif start.next\_node==None:

return

#here string does not match to a node so we transverse though the child node tree

else:

for i in range(len(start.next\_node)):

addNode(start.next\_node[i],string,list1,colNo)

#helps to get sub-attribute

def getSubArgs(listdata,column):

temp=[]

check=0

for i in range(len(listdata)):

check=0

if i==0:

temp.append(listdata[i][column])

for j in range(len(temp)):

if temp[j]==listdata[i][column] :

check=1

if check==0:

temp.append(listdata[i][column])

return temp

#calculates entropy

def entropy(p,n):

if p==0 or n==0 :

return 0.0

else:

total=float(p+n)

ptemp=float(p/total)

ntemp=float(n/total)

logp=float(ptemp\*math.log(ptemp,2))

logn=float(ntemp\*math.log(ntemp,2))

return float(-logp-logn)

#def starts

def desisonTree(list1,string):

#initializing gain list and travelling through each columns

gain=[]

for i in range(0,len(list1[0])-1):

p=0

n=0

I=0

E=0

pin=0

nin=0

G=0

#for each column we find entropy of all rows

for j in range(len(list1)):

# print(list1[j][len(list1[0])-1])

if list1[j][len(list1[0])-1]==1:

p=p+1

else:

n=n+1

I=entropy(p,n)

# finding entropy of each sub-attritute eg. age=old,new and adding it

for k in range(len(getSubArgs(list1,i))):

pin=0

nin=0

for l in range(len(list1)):

if list1[l][i]==getSubArgs(list1,i)[k] and list1[l][len(list1[0])-1]==1:

pin+=1

elif list1[l][i]==getSubArgs(list1,i)[k] and list1[l][len(list1[0])-1]==0:

nin+=1

E=E+float(float(pin+nin)/float(p+n))\*entropy(pin,nin)

#calculating gain

G=float(I)-float(E)

gain.append((i,listcat[i],G))

# if maximum gain is 0 that means that

#it is either completly true or false so directly adding it

if maximum(gain)[2]==0.0:

addNode(head,string,list1,len(list1[0])-1)

else:

# adding maximum gain to tree

addNode(head,string,list1,maximum(gain)[0])

# creating sub list to feed in recursion

for i in range(len(getSubArgs(list1,maximum(gain)[0]))):

newlist=[]

for x in range(len(list1)):

if list1[x][maximum(gain)[0]]==getSubArgs(list1,maximum(gain)[0])[i]:

row=[]

for y in range(1,len(list1[x])):

row.append(list1[x][y])

newlist.append(row)

#calling recurrion

# print("recall")

desisonTree(newlist,getSubArgs(list1,maximum(gain)[0])[i])

desisonTree(datalist,None)

# printTree(head)

print("prediction is that "+str(predict(['new', 'no' ,'software'],head)))

# print(head.next\_node[2].next\_node[0].data)

**OUTPUT:**

d50112@d50112-ThinkCentre-M720t:~/Desktop$ python3 dt.py

prediction is that 1