Experiment 4- ML_LAB

Name - Vishruth Khare

Roll Number - 2K18/CO/393

What is C4.5 Algorithm of Decision Tree?

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set {\displaystyle S={s_{1},s_{2},...}}S={s_{1},s_{2},...} of already classified samples. Each sample {\displaystyle s_{i}}s_{i} consists of a p-dimensional vector {\displaystyle (x_{1,i},x_{2,i},...,x_{p,i})}(x_{{1,i}},x_{{2,i}},...,x_{{p,i}}), where the {\displaystyle x_{j}}x_{j} represent attribute values or features of the sample, as well as the class in which {\displaystyle s_{i}}s_{i} falls.

At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the partitioned sublists.

Importing Libraries

```
import numpy as np
import pprint
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
```

Define the dataset

```
#data in raw form
outlook = 'overcast,overcast,overcast,overcast,rainy,rainy,rainy,rainy,rainy,sunny,sunny,sunn
temp = 'hot,cool,mild,hot,mild,cool,cool,mild,mild,hot,hot,mild,cool,mild'.split(',')
humidity = 'high,normal,high,normal,high,normal,normal,normal,high,high,high,high,normal,norm
windy = 'FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,TRUE'.split
play = 'yes,yes,yes,yes,yes,yes,no,yes,no,no,no,no,yes,yes'.split(',')
```

PROCESSING DATA

```
#conversion into dataset
```

```
dataset ={'outlook':outlook,'temp':temp,'humidity':humidity,'windy':windy,'play':play}

df = pd.DataFrame(dataset,columns=['outlook','temp','humidity','windy','play'])

print(df)
```

```
     outlook  temp humidity  windy play
0   overcast   hot     high  FALSE  yes
1   overcast  cool   normal   TRUE  yes
2   overcast  mild     high   TRUE  yes
3   overcast   hot   normal  FALSE  yes
4      rainy  mild     high  FALSE  yes
5      rainy  cool   normal  FALSE  yes
6      rainy  cool   normal   TRUE   no
7      rainy  mild   normal  FALSE  yes
8      rainy  mild     high   TRUE   no
9      sunny   hot     high  FALSE   no
10     sunny   hot     high   TRUE   no
11     sunny  mild     high  FALSE   no
12     sunny  cool   normal  FALSE  yes
13     sunny  mild   normal   TRUE  yes
```

CALCULATING ENTROPY OF DEPENDENT VARIABLE

```
#calculating entropy of dependent variable
def find_entropy(df):

    Class = df.keys()[-1]

    entropy = 0
    values = df[Class].unique()

    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)

    return entropy
```

CALCULATING INFORMATION GAIN FOR FINDING THE BEST SPLIT AMONG ALL ATTRIBUTES

```
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'
  target_variables = df[Class].unique()  #This gives all 'Yes' and 'No'
  variables = df[attribute].unique()    #This gives different features in that attribute (lik
  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)
```

Function to find Split Info

```
def findSplitInfo(df,Class):
    entr=0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entr += -fraction*np.log2(fraction)
    return entr+0.00000001
```

Double-click (or enter) to edit

COMPARING INFORMATION GAIN OF ALL ATTRIBUTES AND DECIDING THE BEST SPLIT

```
def find_winner(df):

    Entropy_att = []
    IG = []

    for key in df.keys()[:-1]:
#   Entropy_att.append(find_entropy_attribute(df,key))
        infogain = find_entropy(df)-find_entropy_attribute(df,key)
        splitinfo = findSplitInfo(df,key)
#         print(key+ " " + str(splitinfo) +  " " + str(infogain))
        gainratio = infogain/splitinfo
        IG.append(gainratio)
    return df.keys()[:-1][np.argmax(IG)]
```

MAIN RECURSIVE FUNCTION FOR BUILDING SUBTREE AT EACH LEVEL

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


def buildTree(df,tree=None):
    Class = df.keys()[-1]

    node = find_winner(df)

    attValue = np.unique(df[node])


    if tree is None:
        tree={}
        tree[node] = {}


    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
```

Displaying The Tree

```
t =buildTree(df);

import pprint
pprint.pprint(t)
```

```
{'outlook': {'overcast': 'yes',
             'rainy': {'windy': {'FALSE': 'yes', 'TRUE': 'no'}},
             'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}
```