Experiment 5- ML_LAB

Name - Vishruth Khare

Roll Number - 2K18/CO/393

What is Cart Algorithm of Decision Tree?

CART (Classification And Regression Tree) is a decision tree algorithm variation. Decision Trees is the non-parametric supervised learning approach. CART can be applied to both regression and classification problems. This uses gini index as a parameter while splitting.

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

```
#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

```
#converison 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
```

CALCULATING ENTROPY OF DEPENDENT VARIABLE

```
5       rainy  cool   normal  FALSE  yes
```

```
#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

dependent_variable_entropy = find_entropy(df)
#printing entropy of play
print(dependent_variable_entropy)
```

```
0.9402859586706309
```

CALCULATING INFORMATION GAIN FOR FINDING THE BEST SPLIT AMONG ALL ATTRIBUTES

```
#calculating entropy for subtree

def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]
  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)

attribute_list = df.keys()[1]
#printing the column
print(attribute_list)

entropy_temp= find_entropy_attribute(df,attribute_list)
print(entropy_temp)
```

```
temp
0.9110633930116756
```

```
def info_split (df , attribute):

    variables = df[attribute].unique()
    Class = df.keys()[-1]
    target_variables = df[Class].unique()

    ans = 0

    l = len(df)
    for variable in variables:
        l = len(df[attribute][df[attribute]==variable])
        sumratio= 0
        for t in target_variables:
            l1 = len(df[attribute][df[attribute]==variable][df[Class]==t])
            sumratio += (l1/l)**2
        ans += (1-sumratio)*(l/len(df))


    return  ans

attribute_list = df.keys()[2]
entropy_temp= info_split(df,attribute_list)
print(entropy_temp)
```

```
0.3673469387755103
```

COMPARING INFORMATION GAIN OF ALL ATTRIBUTES AND DECIDING THE BEST SPLIT

```
def find_best_split(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
#         Entropy_att.append(find_entropy_attribute(df,key))
        IG.append(info_split(df,key))
    return df.keys()[:-1][np.argmin(IG)]

#finding the root of the tree
root = find_best_split(df)
print(root)
```

```
outlook
```

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_best_split(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:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)

    return tree



Decision_tree = buildTree(df)
pprint.pprint(Decision_tree)
```

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