EXPERIMENT-III

NAME: VISHRUTH KHARE
ROLL NUMBER: 2K18/CO/393

ID3 ALGORITHM

In simple words, a decision tree is a structure that contains nodes (rectangular boxes) and edges(arrows) and is built from a dataset (table of columns representing features/attributes and rows corresponds to records).

Each node is either used to make a decision (known as decision node) or represent an outcome (known as leaf node).

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.

Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.

Most generally ID3 is only used for classification problems with nominal features only.

IMPORTING THE LIBRARIES

In [55]:
```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
import pprint
```

Define the dataset

In [36]:
```
outlook = 'overcast,overcast,overcast,overcast,rainy,rainy,rainy,rainy
temp = 'hot,cool,mild,hot,mild,cool,cool,mild,mild,hot,hot,mild,cool,m
humidity = 'high,normal,high,normal,high,normal,normal,normal,high,hig
windy = 'FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,
play = 'yes,yes,yes,yes,yes,yes,no,yes,no,no,no,no,yes,yes'.split(',')
```

Create Panda Dataframe

In [37]:
```
dataset ={'outlook':outlook,'temp':temp,'humidity':humidity,'windy':wi
df = pd.DataFrame(dataset,columns=['outlook','temp','humidity','windy'
```

In [38]:
```
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
```

Function for finding entropy

In [49]:
```
def find_entropy(df):
    Class = df.keys()[-1]   #To make the code generic, changing target
    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
```

Function for finding InfoGain

In [50]:
```
def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]   #To make the code generic, changing target v
  target_variables = df[Class].unique()  #This gives all 'Yes' and 'No
  variables = df[attribute].unique()    #This gives different features
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][df[Class] =
          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)
```

Largest Info Gain

In [51]:
```
def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
#         Entropy_att.append(find_entropy_attribute(df,key))
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]
```

Function of getting subtable

In [52]:
```
def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)
```

Building tree

In [56]:
```
def buildTree(df,tree=None):
    Class = df.keys()[-1]   #To make the code generic, changing target
    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['Eat'],return_counts=True)
        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) #Calling the funct
    return tree
```

FINAL OUTPUT

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

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