Desh Iyer | 500081889 | Experiment - 4

April 2, 2023

1 Experiment 4 - Implementing a Decision Tree from Scratch using the ID3 Algorithm

In the ID3 algorithm, decision trees are calculated using the concept of entropy and information gain.

[ ]:
```
import pandas as pd
import numpy as np

# eps for making value a bit greater than 0 later on
eps = np.finfo(float).eps

from numpy import log2 as log
```

Creating a dataset,

[ ]:
```
dataset = {'Taste':
 ↪['Salty','Spicy','Spicy','Spicy','Spicy','Sweet','Salty','Sweet','Spicy','Salty'],
       'Temperature':
 ↪['Hot','Hot','Hot','Cold','Hot','Cold','Cold','Hot','Cold','Hot'],
       'Texture':
 ↪['Soft','Soft','Hard','Hard','Hard','Soft','Soft','Soft','Soft','Hard'],
       'Eat':['No','No','Yes','No','Yes','Yes','No','Yes','Yes','Yes']}
```

[ ]:
```
df = pd.DataFrame(dataset,columns=['Taste','Temperature','Texture','Eat'])
df
```

[ ]:

	Taste	Temperature	Texture	Eat
0	Salty	Hot	Soft	No
1	Spicy	Hot	Soft	No
2	Spicy	Hot	Hard	Yes
3	Spicy	Cold	Hard	No
4	Spicy	Hot	Hard	Yes
5	Sweet	Cold	Soft	Yes
6	Salty	Cold	Soft	No
7	Sweet	Hot	Soft	Yes
8	Spicy	Cold	Soft	Yes

```
9  Salty          Hot     Hard  Yes
```

```
[ ]: def find_entropy(df):
         '''
         Function to calculate the entropy of a label
         '''
         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
```

```
[ ]: def find_entropy_attribute(df,attribute):
         '''
         Function to calculate the entropy of all features.
         '''
         Class = df.keys()[-1]
         target_variables = df[Class].unique()
         variables = df[attribute].unique()
         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)
```

```
[ ]: def find_winner(df):
         '''
         Function to find the feature with the highest information gain.
         '''
         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)]
```

```
[ ]: def get_subTable(df, node, value):
         '''
         Function to get a subTable of met conditions.
```

```
    node: Column name
    value: Unique value of the column
    '''
    return df[df[node] == value].reset_index(drop=True)
```

[ ]:
```
def buildTree(df,tree=None):
    '''
    Function to build the ID3 Decision Tree.
    '''
    Class = df.keys()[-1]
    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)

    #Get distinct value of that attribute e.g Salary is node and Low,Med and␣
 ↪High are values
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

   #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    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 function␣
 ↪recursively

    return tree
```

[ ]:
```
tree = buildTree(df)
```

The tree splits are as follows,

[ ]:
```
import pprint
pprint.pprint(tree)
```

```
{'Taste': {'Salty': {'Texture': {'Hard': 'Yes', 'Soft': 'No'}},
           'Spicy': {'Temperature': {'Cold': {'Texture': {'Hard': 'No',
                                                          'Soft': 'Yes'}},
                                     'Hot': {'Texture': {'Hard': 'Yes',
                                                         'Soft': 'No'}}}},
           'Sweet': 'Yes'}}
```

Now, for prediction we go through each node of the tree to find the output.

[ ]:
```
def predict(inst,tree):
    '''
    Function to predict for any input variable.
    '''
    # Recursively we go through the tree that we built earlier

    for nodes in tree.keys():

        value = inst[nodes]
        tree = tree[nodes][value]
        prediction = 0

        if type(tree) is dict:
            prediction = predict(inst, tree)
        else:
            prediction = tree
            break;

    return prediction
```

[ ]:
```
data = {'Taste':'Salty','Temperature':'Cold','Texture':'Hard'}
```

[ ]:
```
inst = pd.Series(data)
```

[ ]:
```
prediction = predict(inst,tree)
prediction
```

[ ]: 'Yes'