In [101]:

```
import numpy as np
import pandas as pd
import math
from pathlib import Path
from pprint import pprint
```

In [102]:

```
df_test1 = pd.read_csv("test_set1.csv")
df_train1 = pd.read_csv("training_set1.csv")
df_validation1 = pd.read_csv("validation_set1.csv")

df_test2 = pd.read_csv("test_set2.csv")
df_train2 = pd.read_csv("training_set2.csv")
df_validation2 = pd.read_csv("validation_set2.csv")
```

In [103]:

```
data = df_train1.values
data
```

Out[103]:

```
array([[1, 0, 0, ..., 1, 1, 1],
       [1, 1, 0, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 1, 1],
       ...,
       [0, 1, 1, ..., 1, 0, 0],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [ ]:

```

```

In [104]:

```
def Freq(data):

    unique,counts = np.unique(data[:,-1],return_counts=True)

    freq = unique[counts.argmax()]

    return freq
```

In [105]:

```
def cal_entropy(data):

    _,counts = np.unique(data[:,-1],return_counts = True)
    prob = counts/counts.sum()
    entropy = sum(prob * -np.log2(prob))

    return entropy
```

In [106]:

```
def attributeEntropy(data_below,data_above):

    p1 = len(data_below) /(len(data_below) + len(data_above))
    p2 = len(data_above) / (len(data_below) + len(data_above))

    attr_entropy = (p1*cal_entropy(data_below) + p2*cal_entropy(data

    return attr_entropy
```

In [107]:

```
 def check_purity(data):

        if len(np.unique(data[:,-1])) !=1:
            return False
        else:
            return True
```

In [108]:

```
def potential_splits1(data):

    potential_attrs = {}
    _,cols = data.shape
    for i in range(cols-1):

        if len(np.unique(data[:,i])) > 1:
            potential_attrs[i] = np.unique(data[:,i])

    return potential_attrs
```

In [109]:

```
def split_set(data,best_attribute,bestattr_value):

    values = data[:,best_attribute]

    arr1 = data[values == bestattr_value]
    arr2 = data[values != bestattr_value]

    return arr1, arr2
```

In [110]:

```
def best_split(data):
    if len(data[:, -1]) == 0:    # empty data
        split = 0

    else:
        prediction = np.mean(data[:, -1])
        split = np.mean(((data[:, -1]) - prediction) **2)

    return split
```

In [111]:

```
def divideAttr(data, attrs):

    nodeInformation = len(data)*cal_entropy(data)
    information_gain = -math.inf

    for i in attrs:
        for j in attrs[i]:
            data_below,data_above = split_set(data, best_attribute =
            attrEnt = attributeEntropy(data_below, data_above)

            if ((nodeInformation - attrEnt) > information_gain):
                information_gain = nodeInformation - attrEnt
                best_attribute = i
                best_value = j

    return best_attribute, best_value
```

In [112]:

```
def decision_tree(df,c=0,max_depth=5):


        if c != 0:
            data = df

        else:
            global attr
            attr = df.columns
            data = df.values

        if(check_purity(data)):
            classification = Freq(data)

            return classification

            pprint(tree(df,c))
        else:
            c += 1
            potential_splits = potential_splits1(data)
            split_column, split_value = divideAttr(data, potential_s
            arr1, arr2 = split_set(data, split_column, split_value)
            feature_name = attr[split_column]
            node = "{} = {}".format(feature_name, split_value)
            sub_tree = {node: []}
            left = decision_tree(arr1, c)
            right = decision_tree(arr2, c)
            if left == right:
                sub_tree = right
            else:
                sub_tree[node].append(left)
                sub_tree[node].append(right)
            return sub_tree
```

In [ ]:

```
```

In [113]:

```
tree = decision_tree(df_train1,c =0)
pprint(tree)
```

```
{'XI = 0': [{'XH = 0': [0,
                        {'XB = 0': [0,
                                    {'XN = 0': [0,
                                                {'XK =
0': [0,

{'XC = 0': [1,

0]}]}]}]}]},
            {'XH = 0': [{'XP = 0': [0,
                                    {'XT = 0': [0,
                                                {'XG =
0': [{'XK = 0': [1, 0]},

0]}]}]},
                        {'XC = 0': [{'XN = 0': [{'XD =
0': [{'XP = 0': [{'XO = 0': [{'XK = 0': [0,

1]},

1]},

0]},

{'XU = 0': [{'XG = 0': [0,

1]},

0]}]},
                                                {'XB =
0': [{'XL = 0': [0,

{'XR = 0': [{'XJ = 0': [1,

{'XF = 0': [0,

1]}]},

{'XM = 0': [0,

{'XE = 0': [1,
```

```
0]}]}]}]},

{'XQ = 0': [1,

{'XF = 0': [0,

{'XD = 0': [{'XM = 0': [0,

1]},

1]}]}]}]},
                                        {'XB = 0': [{'XG =
0': [{'XU = 0': [{'XK = 0': [1,

0]},

0]},

0]},
                                                    0]}]}]
}]}
```

In [114]:

```
def evaluate_test(test, tree):
    node = list(tree.keys())[0]
    attribute_name, operator, value = node.split(" ")

    if str(test[attribute_name]) == value:
        answer = tree[node][0]
    else:
        answer = tree[node][1]

    if not isinstance(answer, type(dict)):
        return answer
    else:
        sub_tree = answer
        return evaluate_test(test, sub_tree)
```

In [115]:

```
evaluate_test(df_test1,tree)
```

Out[115]:

```
{'XH = 0': [{'XP = 0': [0,
    {'XT = 0': [0, {'XG = 0': [{'XK = 0': [1, 0]}, 0]}
]}]},
  {'XC = 0': [{'XN = 0': [{'XD = 0': [{'XP = 0': [{'XO
= 0': [{'XK = 0': [0,
                1]},
              1]},
            0]},
          {'XU = 0': [{'XG = 0': [0, 1]}, 0]}]},
        {'XB = 0': [{'XL = 0': [0,
            {'XR = 0': [{'XJ = 0': [1, {'XF = 0': [0, 1]
}]},
              {'XM = 0': [0, {'XE = 0': [1, 0]}]}]}]},
          {'XQ = 0': [1,
            {'XF = 0': [0, {'XD = 0': [{'XM = 0': [0, 1]
}, 1]}]}]}]}]},
    {'XB = 0': [{'XG = 0': [{'XU = 0': [{'XK = 0': [1,
0]}, 0]}, 0]}, 0]}]}]}
```

In [116]:

```
evaluate_test(df_validation1,tree)
```

Out[116]:

```
{'XH = 0': [{'XP = 0': [0,
    {'XT = 0': [0, {'XG = 0': [{'XK = 0': [1, 0]}, 0]}
]}]},
  {'XC = 0': [{'XN = 0': [{'XD = 0': [{'XP = 0': [{'XO
= 0': [{'XK = 0': [0,
                1]},
              1]},
            0]},
          {'XU = 0': [{'XG = 0': [0, 1]}, 0]}]},
      {'XB = 0': [{'XL = 0': [0,
          {'XR = 0': [{'XJ = 0': [1, {'XF = 0': [0, 1]
}]},
            {'XM = 0': [0, {'XE = 0': [1, 0]}]}]}]}]},
        {'XQ = 0': [1,
          {'XF = 0': [0, {'XD = 0': [{'XM = 0': [0, 1]
}, 1]}]}]}]}]},
    {'XB = 0': [{'XG = 0': [{'XU = 0': [{'XK = 0': [1,
0]}, 0]}, 0]}, 0]}]}]}
```

In [117]:

```
def accuracy(df,tree):

    df["prediction"] = df.apply(evaluate_test, args=(tree,), axis =
    df["correct_prediction"] = df["prediction"] == df["Class"]
    accuracy = df["correct_prediction"].mean()

    return accuracy
```

In [118]:

```
accuracy = accuracy(df_test1,tree)
accuracy
```

Out[118]:

```
0.0
```

In [ ]:

In [101]:

In [102]:

In [103]:

Out[103]:

```
array([[1, 0, 0, ..., 1, 1, 1],
       [1, 1, 0, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 1, 1],
       ...,
       [0, 1, 1, ..., 1, 0, 0],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [ ]:

In [104]:

In [105]:

In [106]:

In [107]:

In [108]:

In [109]:

In [110]:

In [111]:

In [112]:

In [ ]:

In [113]:

```
{'XI = 0': [{'XH = 0': [0,
                        {'XB = 0': [0,
                                    {'XN = 0': [0,
                                                {'XK =
0': [0,

{'XC = 0': [1,

0]}]}]}]}]},
            {'XH = 0': [{'XP = 0': [0,
                                    {'XT = 0': [0,
                                                {'XG =
0': [{'XK = 0': [1, 0]},

0]}]}]},
                        {'XC = 0': [{'XN = 0': [{'XD =
0': [{'XP = 0': [{'XO = 0': [{'XK = 0': [0,

1]},
```

In [114]:

In [115]:

Out[115]:

```
{'XH = 0': [{'XP = 0': [0,
    {'XT = 0': [0, {'XG = 0': [{'XK = 0': [1, 0]}, 0]}
]}]},
  {'XC = 0': [{'XN = 0': [{'XD = 0': [{'XP = 0': [{'XO
= 0': [{'XK = 0': [0,
                1]},
              1]},
            0]},
          {'XU = 0': [{'XG = 0': [0, 1]}, 0]}]},
       {'XB = 0': [{'XL = 0': [0,
           {'XR = 0': [{'XJ = 0': [1, {'XF = 0': [0, 1]
}]},
             {'XM = 0': [0, {'XE = 0': [1, 0]}]}]}]},
          {'XQ = 0': [1,
           {'XF = 0': [0, {'XD = 0': [{'XM = 0': [0, 1]
}, 1]}]}]}]}]},
    {'XB = 0': [{'XG = 0': [{'XU = 0': [{'XK = 0': [1,
0]}, 0]}, 0]}, 0]}]}]}
```

In [116]:

Out[116]:

```
{'XH = 0': [{'XP = 0': [0,
    {'XT = 0': [0, {'XG = 0': [{'XK = 0': [1, 0]}, 0]}
]}]},
  {'XC = 0': [{'XN = 0': [{'XD = 0': [{'XP = 0': [{'XO
= 0': [{'XK = 0': [0,
                1]},
              1]},
            0]},
          {'XU = 0': [{'XG = 0': [0, 1]}, 0]}]},
      {'XB = 0': [{'XL = 0': [0,
          {'XR = 0': [{'XJ = 0': [1, {'XF = 0': [0, 1]
}]},
            {'XM = 0': [0, {'XE = 0': [1, 0]}]}]}]},
        {'XQ = 0': [1,
          {'XF = 0': [0, {'XD = 0': [{'XM = 0': [0, 1]
}, 1]}]}]}]}]},
    {'XB = 0': [{'XG = 0': [{'XU = 0': [{'XK = 0': [1,
0]}, 0]}, 0]}, 0]}]}]}
```

In [117]:

In [118]:

Out[118]:

```
0.0
```

In [ ]: