Task 2: [1] Implement a decision tree (DT) classifier by using one the information gain as the split criteria.

Name: Yeo Zheng Xu Isaac

In [1]:
```
import pandas as pd
import numpy as np

# Read in the CSV file
# convert all values "?" to null
df = pd.read_csv("C:/Users/Isaac Yeo/Downloads/magic04.data",
                  header=None, na_values="?" )
df.head()
```

Out[1]:

	0	1	2	3	4	5	6	7	8	9	1
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	

In [2]:
```
df[10].value_counts()
```

Out[2]:
```
g    12332
h     6688
Name: 10, dtype: int64
```

In [3]:
```
cleanup_nums = {10: {"g": 0, "h": 1}}
df.replace(cleanup_nums, inplace=True)
df.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	1
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	

In [4]:
```
dt=np.array(df)
dt
```

Out[4]:
```
array([[ 28.7967,  16.0021,   2.6449, ...,  40.092 ,  81.8828,   0.    ],
       [ 31.6036,  11.7235,   2.5185, ...,   6.3609, 205.261 ,   0.    ],
       [162.052 , 136.031 ,   4.0612, ...,  76.96  , 256.788 ,   0.    ],
       ...,
       [ 75.4455,  47.5305,   3.4483, ...,  30.2987, 256.5166,   1.    ],
       [120.5135,  76.9018,   3.9939, ...,  84.6874, 408.3166,   1.    ],
       [187.1814,  53.0014,   3.2093, ...,  52.731 , 272.3174,   1.    ]])
```

Decision Tree Implementation as follows below:

STEPS & FUNCTIONS EXPLAINATION FOR DECISION TREE IMPLEMENTATION BELOW:

[1] Create Split:

Split refers to the attribute and the value that helps to divide the attribute index rows. This is achieved by finding gini index by 3 bits, breaking and evaluating the dataset.

[2] Gini index:

To test the "split" dataset by separating the training patterns of two row groups and ini index help() to find the data set's gini index.

[3] Split dataset:

This function breaks the dataset into two list of attribute and value indexed rows. After that, we can use the gini index to evaluate the cost of failed test split() by checking that of all attribute value that belongs either to the left or right group.

[4] Evaluating:

Use both above functions to find the best split, so then we can use it as a node. get split() is used to find the best split documented by iterating each attribute and values and returning them after checks are done.

[5] Build Tree:

Build tree() function is to create the root node and the entire tree by calling the spilt() function which helps to use both the node as an argument and the max depth size of the node.

In [25]:
```
from random import seed
from random import randrange
from csv import reader

### -- Split a dataset based on [1] attribute & [2] attribute value -- ###
def testing_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right

### -- Calculate Gini index for split dataset  -- ###
def gini_index(groups, classes):
   ### -- count all samples at split point  -- ###
    n_instances = float(sum([len(group) for group in groups]))
 ### -- sum weighted Gini index for each group  -- ###
    gini = 0.0
    for group in groups:
        size = float(len(group))
        if size == 0:
            continue
        score = 0.0
      ### -- score the group based on the score for each class  -- ###
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
      ### -- weigh the group score by its relative size  -- ###
        gini += (1.0 - score) * (size / n_instances)
    return gini

### -- Select the best split point for a dataset  -- ###
def getData_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = testing_split(index, row[index], dataset)
            gini = gini_index(groups, class_values)
            if gini < b_score:
                b_index, b_value, b_score, b_groups = index, row[index], gini,
groups
    return {'index':b_index, 'value':b_value, 'groups':b_groups}

### -- Create a terminal node value  -- ###
def to_terminal(group):
    outcomes = [row[-1] for row in group]
    return max(set(outcomes), key=outcomes.count)

### -- Create child splits for a node or make terminal  -- ###
def split(node, max_depth, min_size, depth):
    left, right = node['groups']
    del(node['groups'])
  ### -- Code is to check for a no split  -- ###
```

```
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
  ### -- Code is to check for max depth  -- ###
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left), to_terminal(right)
        return
 ### -- To process left child  -- ###
    if len(left) <= min_size:
        node['left'] = to_terminal(left)
    else:
        node['left'] = getData_split(left)
        split(node['left'], max_depth, min_size, depth+1)
  ### -- To process right child  -- ###
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    else:
        node['right'] = getData_split(right)
        split(node['right'], max_depth, min_size, depth+1)

# Build a decision tree
def build_tree(train, max_depth, min_size):
    root = getData_split(train)
    split(root, max_depth, min_size, 1)
    return root

### -- Print function for decision tree  -- ###
def print_decision_tree(node, depth=0):
    if isinstance(node, dict):
        print('%s[X%d < %.3f]' % ((depth*' ', (node['index']+1), node['value'
])))
        print_tree(node['left'], depth+1)
        print_tree(node['right'], depth+1)
    else:
        print('%s[%s]' % ((depth*' ', node)))
```

In [28]:
```
Deci_tree =build_tree(dt,10,1)
print_decision_tree(tree)
```

```
                   [X9 < 26.298]
                    [X1 < 117.868]
                     [X9 < 11.822]
                      [X7 < -71.845]
                       [X2 < 41.873]
                        [X9 < 2.759]
                         [X1 < 109.370]
                          [X1 < 70.437]
                           [0.0]
                           [0.0]
                          [X1 < 109.370]
                           [0.0]
                           [0.0]
                         [X3 < 2.797]
                          [X1 < 76.805]
                           [1.0]
                           [X1 < 89.596]
                            [X1 < 76.805]
                             [0.0]
                             [0.0]
                            [X1 < 89.596]
                             [0.0]
                             [0.0]
                          [X4 < 0.290]
                           [X10 < 271.107]
                            [X4 < 0.256]
                             [1.0]
                             [0.0]
                            [X7 < -83.503]
                             [1.0]
                             [0.0]
                           [X1 < 90.939]
                            [1.0]
                            [X1 < 90.939]
                             [1.0]
                             [1.0]
                        [X9 < 0.369]
                         [0.0]
                         [X8 < 53.439]
                          [X1 < 98.811]
                           [X1 < 85.750]
                            [X1 < 82.474]
                             [1.0]
                             [1.0]
                            [X1 < 85.750]
                             [1.0]
                             [1.0]
                           [X1 < 98.811]
                            [1.0]
                            [1.0]
                          [X1 < 115.325]
                           [1.0]
                           [0.0]
                       [X5 < 0.298]
                        [X2 < 57.592]
                         [X2 < 9.166]
                          [X3 < 2.597]
```

```
          [X9 < 6.497]
           [X1 < 27.293]
            [1.0]
            [0.0]
           [X5 < 0.257]
            [1.0]
            [0.0]
          [X1 < 101.658]
           [X7 < -18.843]
            [0.0]
            [1.0]
           [0.0]
         [X10 < 39.231]
          [X1 < 73.664]
           [X2 < 15.440]
            [1.0]
            [0.0]
           [X1 < 79.161]
            [1.0]
            [1.0]
          [X7 < -47.929]
           [X9 < 4.420]
            [0.0]
            [0.0]
           [X4 < 0.325]
            [0.0]
            [0.0]
        [X3 < 4.182]
         [X7 < 110.795]
          [X8 < -83.627]
           [X1 < 90.867]
            [1.0]
            [0.0]
           [X1 < 65.832]
            [1.0]
            [1.0]
          [0.0]
         [X1 < 114.722]
          [X1 < 106.821]
           [X1 < 101.886]
            [0.0]
            [0.0]
           [X1 < 106.821]
            [0.0]
            [0.0]
          [X1 < 114.722]
           [0.0]
           [0.0]
       [X3 < 2.433]
        [X1 < 17.941]
         [X2 < 10.683]
          [X4 < 0.777]
           [X5 < 0.512]
            [1.0]
            [0.0]
           [X3 < 2.302]
            [0.0]
```

```
                [1.0]
              [X6 < -16.096]
               [X1 < 12.345]
                [1.0]
                [1.0]
               [X8 < -11.786]
                [1.0]
                [0.0]
             [X2 < 13.343]
              [X7 < 25.338]
               [X1 < 32.401]
                [0.0]
                [0.0]
               [X7 < 27.249]
                [1.0]
                [0.0]
              [X3 < 2.321]
               [X1 < 25.986]
                [1.0]
                [1.0]
               [X2 < 13.352]
                [1.0]
                [0.0]
            [X4 < 0.587]
             [X3 < 2.616]
              [X1 < 17.415]
               [X1 < 17.060]
                [1.0]
                [1.0]
               [X2 < 8.511]
                [1.0]
                [0.0]
              [X4 < 0.481]
               [X2 < 19.087]
                [0.0]
                [1.0]
               [X4 < 0.526]
                [1.0]
                [1.0]
             [X6 < -13.681]
              [X4 < 0.616]
               [X1 < 49.044]
                [1.0]
                [1.0]
               [X4 < 0.756]
                [0.0]
                [1.0]
              [X5 < 0.346]
               [X4 < 0.608]
                [1.0]
                [0.0]
               [X2 < 15.833]
                [1.0]
                [0.0]
         [X1 < 65.438]
          [X2 < 10.551]
           [X3 < 2.447]
```

```
[X1 < 17.497]
 [X4 < 0.788]
  [X5 < 0.523]
   [X1 < 9.441]
    [1.0]
    [1.0]
   [X1 < 12.983]
    [1.0]
    [0.0]
  [X1 < 12.891]
   [X1 < 10.329]
    [1.0]
    [1.0]
   [X3 < 2.397]
    [0.0]
    [1.0]
 [X4 < 0.810]
  [X8 < 13.291]
   [X8 < -17.327]
    [1.0]
    [0.0]
   [X1 < 22.446]
    [1.0]
    [1.0]
  [X5 < 0.521]
   [X10 < 80.059]
    [1.0]
    [0.0]
   [X1 < 19.792]
    [1.0]
    [1.0]
[X3 < 2.523]
 [X1 < 21.004]
  [X10 < 207.077]
   [X1 < 18.185]
    [1.0]
    [1.0]
   [X2 < 10.414]
    [1.0]
    [0.0]
  [X2 < 7.180]
   [X1 < 31.655]
    [1.0]
    [1.0]
   [X6 < 7.938]
    [0.0]
    [0.0]
 [X3 < 2.623]
  [X5 < 0.270]
   [X10 < 187.954]
    [0.0]
    [1.0]
   [X9 < 14.226]
    [0.0]
    [1.0]
  [X9 < 14.323]
   [X9 < 14.238]
```

```
        [1.0]
        [0.0]
       [X1 < 53.586]
        [1.0]
        [1.0]
    [X10 < 239.062]
     [X7 < -27.440]
      [X9 < 22.683]
       [X2 < 24.937]
        [X2 < 12.636]
         [1.0]
         [0.0]
        [X6 < -40.374]
         [0.0]
         [1.0]
       [X8 < -14.097]
        [X1 < 48.184]
         [0.0]
         [0.0]
        [X1 < 58.714]
         [1.0]
         [1.0]
      [X1 < 47.471]
       [X5 < 0.241]
        [X2 < 11.277]
         [0.0]
         [0.0]
        [X3 < 2.621]
         [0.0]
         [0.0]
       [X2 < 13.381]
        [X3 < 2.712]
         [0.0]
         [1.0]
        [X9 < 20.099]
         [0.0]
         [0.0]
     [X2 < 30.932]
      [X5 < 0.246]
       [X9 < 13.361]
        [X2 < 13.601]
         [1.0]
         [0.0]
        [X3 < 2.649]
         [0.0]
         [0.0]
       [X3 < 2.637]
        [X6 < -0.419]
         [0.0]
         [0.0]
        [X9 < 12.968]
         [0.0]
         [1.0]
      [X3 < 3.200]
       [X3 < 2.804]
        [X1 < 55.362]
         [1.0]
```

```
                    [0.0]
                   [X1 < 58.378]
                    [1.0]
                    [1.0]
                  [X1 < 56.542]
                   [1.0]
                   [X1 < 63.690]
                    [0.0]
                    [0.0]
             [X7 < 49.603]
              [X3 < 2.832]
               [X10 < 152.639]
                [X1 < 92.751]
                 [X1 < 90.709]
                  [X1 < 75.411]
                   [1.0]
                   [1.0]
                  [X1 < 90.709]
                   [1.0]
                   [1.0]
                 [X1 < 94.108]
                  [0.0]
                  [X1 < 94.108]
                   [1.0]
                   [1.0]
                [X7 < -85.402]
                 [X1 < 92.433]
                  [X1 < 75.698]
                   [1.0]
                   [1.0]
                  [X1 < 92.433]
                   [1.0]
                   [1.0]
                 [X9 < 16.755]
                  [X6 < -31.379]
                   [0.0]
                   [1.0]
                  [X10 < 227.657]
                   [0.0]
                   [1.0]
               [X1 < 69.726]
                [X1 < 67.324]
                 [X1 < 65.973]
                  [X1 < 65.554]
                   [1.0]
                   [1.0]
                  [X1 < 65.973]
                   [1.0]
                   [1.0]
                 [X4 < 0.196]
                  [X1 < 67.324]
                   [0.0]
                   [0.0]
                  [X6 < 9.084]
                   [1.0]
                   [0.0]
                [X6 < -150.418]
```

```
      [0.0]
      [X1 < 81.707]
       [X1 < 80.325]
        [1.0]
        [0.0]
       [X10 < 191.537]
        [1.0]
        [1.0]
   [X9 < 20.272]
    [X1 < 103.109]
     [X10 < 311.568]
      [X2 < 12.503]
       [X1 < 90.663]
        [1.0]
        [0.0]
       [X6 < -126.614]
        [1.0]
        [0.0]
      [X9 < 15.807]
       [X2 < 24.138]
        [1.0]
        [0.0]
       [X1 < 97.868]
        [1.0]
        [1.0]
     [X8 < -40.384]
      [X1 < 107.242]
       [0.0]
       [0.0]
      [X3 < 2.718]
       [0.0]
       [X2 < 32.372]
        [1.0]
        [1.0]
    [X7 < 65.037]
     [X1 < 100.278]
      [X1 < 79.374]
       [X1 < 67.447]
        [1.0]
        [1.0]
       [X1 < 79.374]
        [1.0]
        [1.0]
      [X1 < 100.278]
       [1.0]
       [1.0]
     [X1 < 93.423]
      [X8 < -24.319]
       [X1 < 71.131]
        [1.0]
        [1.0]
       [X9 < 24.909]
        [0.0]
        [1.0]
      [X7 < 93.408]
       [X6 < 2.331]
        [1.0]
```

```
            [1.0]
           [X1 < 102.346]
            [0.0]
            [1.0]
   [X9 < 8.699]
    [X2 < 40.208]
     [X10 < 158.745]
      [X1 < 141.751]
       [X1 < 119.664]
        [1.0]
        [1.0]
       [X1 < 141.751]
        [1.0]
        [1.0]
      [X4 < 0.178]
       [X7 < 46.792]
        [X1 < 120.990]
         [0.0]
         [1.0]
        [X1 < 130.346]
         [X1 < 122.510]
          [X1 < 118.862]
           [0.0]
           [0.0]
          [X1 < 122.510]
           [0.0]
           [0.0]
         [X1 < 130.346]
          [0.0]
          [0.0]
       [X2 < 20.402]
        [X3 < 3.242]
         [X1 < 117.986]
          [X1 < 117.927]
           [0.0]
           [1.0]
          [X6 < 89.551]
           [0.0]
           [0.0]
         [1.0]
        [X3 < 2.674]
         [X1 < 125.302]
          [0.0]
          [X1 < 125.302]
           [0.0]
           [0.0]
         [X5 < 0.199]
          [X10 < 262.948]
           [1.0]
           [0.0]
          [X9 < 8.201]
           [1.0]
           [0.0]
     [X4 < 0.100]
      [X7 < 35.931]
       [X7 < -52.087]
        [X1 < 191.162]
```

```
           [X1 < 153.225]
            [X1 < 136.786]
             [1.0]
             [1.0]
            [X1 < 153.225]
             [1.0]
             [1.0]
           [X1 < 191.162]
            [1.0]
            [1.0]
          [X3 < 4.118]
           [X1 < 143.470]
            [1.0]
            [X1 < 143.470]
             [1.0]
             [1.0]
           [0.0]
         [X8 < 39.278]
          [X1 < 138.085]
           [X1 < 133.103]
            [X1 < 121.846]
             [0.0]
             [0.0]
            [0.0]
           [0.0]
          [X3 < 4.616]
           [X1 < 124.014]
            [1.0]
            [1.0]
           [0.0]
        [X1 < 119.000]
         [0.0]
         [X1 < 146.416]
          [X1 < 139.728]
           [X1 < 136.652]
            [X1 < 124.186]
             [1.0]
             [1.0]
            [X1 < 136.652]
             [1.0]
             [1.0]
           [X1 < 139.728]
            [1.0]
            [1.0]
          [X1 < 146.416]
           [1.0]
           [1.0]
      [X7 < -61.959]
       [X1 < 160.068]
        [X1 < 134.999]
         [X1 < 133.339]
          [X1 < 130.885]
           [X1 < 120.543]
            [X1 < 118.539]
             [1.0]
             [1.0]
            [X1 < 120.543]
```

```
                [1.0]
                [1.0]
              [X1 < 130.885]
               [1.0]
               [1.0]
             [X1 < 133.339]
              [1.0]
              [1.0]
            [X1 < 134.999]
             [1.0]
             [1.0]
           [X1 < 160.068]
            [1.0]
            [1.0]
          [X7 < -57.274]
           [X1 < 126.665]
            [0.0]
            [0.0]
           [X3 < 4.803]
            [X5 < 0.120]
             [X5 < 0.090]
              [X1 < 119.261]
               [X1 < 119.038]
                [1.0]
                [0.0]
               [X1 < 124.972]
                [1.0]
                [1.0]
              [X1 < 122.595]
               [X1 < 121.318]
                [1.0]
                [1.0]
               [X1 < 179.439]
                [0.0]
                [1.0]
             [X3 < 2.597]
              [X1 < 147.899]
               [1.0]
               [0.0]
              [X1 < 150.302]
               [X1 < 134.831]
                [1.0]
                [1.0]
               [X1 < 150.302]
                [1.0]
                [1.0]
            [0.0]
       [X1 < 36.261]
        [X3 < 2.331]
         [X1 < 12.355]
          [X4 < 0.847]
           [X1 < 10.140]
            [X1 < 9.954]
             [X1 < 7.208]
              [1.0]
              [1.0]
             [X1 < 9.954]
```

```
    [1.0]
    [1.0]
  [X1 < 10.140]
   [1.0]
   [1.0]
 [0.0]
[X2 < 6.804]
 [X1 < 19.642]
  [X1 < 14.363]
   [X1 < 13.463]
    [1.0]
    [1.0]
   [X1 < 14.363]
    [1.0]
    [1.0]
  [X2 < 0.064]
   [X3 < 2.280]
    [X8 < 19.648]
     [X1 < 21.806]
      [0.0]
      [0.0]
     [1.0]
    [X5 < 0.416]
     [0.0]
     [X1 < 22.130]
      [1.0]
      [1.0]
   [X2 < 5.689]
    [X1 < 23.269]
     [X1 < 22.957]
      [1.0]
      [1.0]
     [X1 < 23.269]
      [1.0]
      [1.0]
    [X2 < 6.782]
     [X3 < 2.193]
      [0.0]
      [0.0]
     [X3 < 2.119]
      [1.0]
      [0.0]
 [X7 < -20.908]
  [X1 < 18.113]
   [X1 < 15.636]
    [1.0]
    [1.0]
   [X1 < 18.113]
    [1.0]
    [1.0]
  [X1 < 28.680]
   [X7 < 20.445]
    [X10 < 220.824]
     [X6 < 33.860]
      [0.0]
      [1.0]
     [X4 < 0.803]
```

```
       [0.0]
       [1.0]
     [X9 < 74.308]
      [X6 < 10.298]
       [1.0]
       [0.0]
      [X1 < 25.569]
       [0.0]
       [0.0]
    [X9 < 56.146]
     [X4 < 0.653]
      [X4 < 0.427]
       [1.0]
       [0.0]
      [X1 < 28.886]
       [1.0]
       [1.0]
     [X10 < 90.303]
      [X1 < 30.196]
       [0.0]
       [0.0]
      [X8 < -8.904]
       [0.0]
       [1.0]
[X2 < 11.281]
 [X3 < 2.454]
  [X2 < 9.388]
   [X9 < 49.589]
    [X1 < 20.206]
     [X5 < 0.581]
      [X10 < 223.884]
       [1.0]
       [0.0]
      [X1 < 13.155]
       [0.0]
       [0.0]
     [X3 < 2.387]
      [X2 < 6.791]
       [1.0]
       [0.0]
      [X9 < 31.218]
       [0.0]
       [1.0]
    [X3 < 2.334]
     [X2 < 7.436]
      [X1 < 18.561]
       [0.0]
       [0.0]
      [1.0]
     [X6 < -27.867]
      [X1 < 28.146]
       [1.0]
       [0.0]
      [X3 < 2.453]
       [1.0]
       [0.0]
   [X10 < 222.430]
```

```
        [X4 < 0.543]
         [X10 < 166.892]
          [X4 < 0.527]
           [0.0]
           [0.0]
          [X2 < 10.213]
           [1.0]
           [0.0]
         [X6 < -2.277]
          [X10 < 175.035]
           [0.0]
           [1.0]
          [X6 < 26.773]
           [0.0]
           [1.0]
        [X3 < 2.365]
         [X3 < 2.350]
          [X7 < -15.924]
           [0.0]
           [1.0]
          [X7 < 5.204]
           [1.0]
           [0.0]
         [X8 < -5.813]
          [X2 < 9.849]
           [0.0]
           [1.0]
          [X1 < 15.876]
           [1.0]
           [1.0]
      [X3 < 2.526]
       [X2 < 9.623]
        [X9 < 38.947]
         [X4 < 0.496]
          [0.0]
          [X9 < 37.469]
           [1.0]
           [0.0]
         [X5 < 0.496]
          [X1 < 14.316]
           [1.0]
           [1.0]
          [X4 < 0.696]
           [0.0]
           [1.0]
        [X2 < 9.746]
         [X8 < 11.350]
          [X1 < 34.424]
           [0.0]
           [0.0]
          [1.0]
         [X10 < 78.764]
          [X8 < 7.290]
           [1.0]
           [0.0]
          [X9 < 32.666]
           [0.0]
```

```
              [1.0]
            [X10 < 37.257]
             [X1 < 31.387]
              [X1 < 31.293]
               [1.0]
               [1.0]
              [0.0]
             [X2 < 10.863]
              [X6 < -24.517]
               [X6 < -24.779]
                [1.0]
                [0.0]
               [X1 < 25.735]
                [1.0]
                [1.0]
              [X2 < 10.867]
               [0.0]
               [X3 < 2.535]
                [0.0]
                [1.0]
          [X10 < 172.685]
           [X9 < 48.305]
            [X7 < -25.888]
             [X10 < 43.695]
              [X1 < 33.429]
               [0.0]
               [0.0]
              [X1 < 35.804]
               [X1 < 25.362]
                [1.0]
                [1.0]
               [0.0]
             [X5 < 0.212]
              [X2 < 11.725]
               [1.0]
               [X4 < 0.414]
                [0.0]
                [1.0]
              [X1 < 33.545]
               [X5 < 0.397]
                [0.0]
                [1.0]
               [X7 < -5.101]
                [0.0]
                [1.0]
            [X2 < 14.731]
             [X3 < 2.526]
              [X1 < 15.021]
               [X1 < 14.419]
                [1.0]
                [1.0]
               [X1 < 30.151]
                [0.0]
                [1.0]
              [X10 < 106.394]
               [X5 < 0.312]
                [0.0]
```

```
        [1.0]
       [X9 < 54.581]
        [1.0]
        [1.0]
     [X10 < 132.570]
      [X1 < 17.104]
       [X1 < 17.090]
        [1.0]
        [1.0]
       [X7 < -25.531]
        [1.0]
        [0.0]
      [X1 < 29.682]
       [X4 < 0.575]
        [0.0]
        [1.0]
       [X5 < 0.142]
        [0.0]
        [1.0]
   [X9 < 41.381]
    [X10 < 207.963]
     [X1 < 18.773]
      [X10 < 205.390]
       [X2 < 15.569]
        [1.0]
        [0.0]
       [X1 < 16.193]
        [0.0]
        [0.0]
      [X3 < 3.139]
       [X4 < 0.344]
        [0.0]
        [0.0]
       [X1 < 34.703]
        [1.0]
        [1.0]
     [X7 < 20.455]
      [X3 < 2.566]
       [X10 < 210.945]
        [1.0]
        [0.0]
       [X10 < 242.883]
        [0.0]
        [1.0]
      [X8 < -11.373]
       [X3 < 2.621]
        [1.0]
        [0.0]
       [X9 < 30.612]
        [1.0]
        [1.0]
    [X3 < 2.489]
     [X1 < 29.709]
      [X4 < 0.521]
       [X9 < 74.229]
        [0.0]
        [1.0]
```

```
       [X10 < 205.180]
        [0.0]
        [1.0]
      [X2 < 12.350]
       [0.0]
       [X1 < 30.111]
        [1.0]
        [1.0]
     [X2 < 16.284]
      [X1 < 19.687]
       [X8 < -6.170]
        [1.0]
        [0.0]
       [X2 < 14.132]
        [1.0]
        [1.0]
      [X1 < 26.742]
       [X6 < 20.720]
        [0.0]
        [1.0]
       [X6 < -30.663]
        [0.0]
        [1.0]
[X1 < 57.484]
 [X9 < 41.631]
  [X2 < 12.251]
   [X3 < 2.435]
    [X1 < 38.305]
     [1.0]
     [X1 < 44.304]
      [X1 < 38.305]
       [0.0]
       [0.0]
      [0.0]
    [X3 < 2.600]
     [X6 < 32.687]
      [X3 < 2.592]
       [X1 < 41.433]
        [1.0]
        [1.0]
       [0.0]
      [X1 < 45.133]
       [X1 < 39.859]
        [0.0]
        [0.0]
       [1.0]
     [X1 < 48.655]
      [X1 < 47.823]
       [X1 < 37.915]
        [1.0]
        [1.0]
       [X1 < 47.823]
        [1.0]
        [1.0]
      [X1 < 48.655]
       [1.0]
       [1.0]
```

```
        [X7 < 11.039]
         [X1 < 41.184]
          [X9 < 29.519]
           [X1 < 36.520]
            [0.0]
            [0.0]
           [X9 < 31.918]
            [X1 < 37.151]
             [1.0]
             [1.0]
            [X1 < 39.661]
             [0.0]
             [0.0]
          [X10 < 180.167]
           [X8 < -12.486]
            [X1 < 47.052]
             [0.0]
             [0.0]
            [X9 < 34.257]
             [1.0]
             [1.0]
           [X1 < 49.572]
            [X1 < 42.370]
             [1.0]
             [1.0]
            [X1 < 49.572]
             [1.0]
             [1.0]
         [X10 < 89.330]
          [X1 < 46.291]
           [X1 < 39.071]
            [X1 < 38.025]
             [0.0]
             [0.0]
            [X1 < 39.071]
             [0.0]
             [0.0]
           [X1 < 46.291]
            [0.0]
            [0.0]
          [X3 < 2.670]
           [X2 < 22.100]
            [X5 < 0.306]
             [0.0]
             [1.0]
            [X1 < 36.922]
             [1.0]
             [1.0]
           [X2 < 17.231]
            [X1 < 38.069]
             [0.0]
             [1.0]
            [X9 < 27.932]
             [0.0]
             [0.0]
       [X10 < 133.370]
        [X2 < 14.526]
```

```
        [X3 < 2.548]
         [X1 < 41.754]
          [1.0]
          [0.0]
         [X1 < 44.728]
          [X1 < 42.094]
           [X1 < 39.091]
            [1.0]
            [1.0]
           [X1 < 42.094]
            [1.0]
            [1.0]
          [X1 < 44.728]
           [1.0]
           [1.0]
        [X5 < 0.191]
         [X1 < 55.958]
          [X6 < 34.030]
           [X7 < 31.225]
            [0.0]
            [0.0]
           [X4 < 0.188]
            [0.0]
            [1.0]
          [X1 < 56.553]
           [X1 < 55.958]
            [1.0]
            [1.0]
           [X1 < 56.553]
            [1.0]
            [1.0]
         [X6 < 26.491]
          [X1 < 42.249]
           [X1 < 41.822]
            [1.0]
            [1.0]
           [X1 < 42.249]
            [1.0]
            [1.0]
          [X2 < 19.553]
           [X1 < 46.207]
            [0.0]
            [0.0]
           [1.0]
       [X3 < 3.336]
        [X10 < 352.882]
         [X2 < 0.388]
          [0.0]
          [X2 < 13.483]
           [X1 < 38.629]
            [1.0]
            [1.0]
           [X2 < 13.502]
            [0.0]
            [1.0]
         [X2 < 18.890]
          [X1 < 42.452]
```

```
          [0.0]
          [0.0]
         [X7 < -21.119]
          [0.0]
          [X1 < 37.802]
           [1.0]
           [1.0]
       [X2 < 37.248]
        [X3 < 3.533]
         [X1 < 37.355]
          [1.0]
          [1.0]
         [0.0]
        [X1 < 56.143]
         [X1 < 43.899]
          [0.0]
          [0.0]
         [X1 < 56.143]
          [0.0]
          [0.0]
    [X3 < 2.840]
     [X7 < -123.359]
      [X3 < 2.679]
       [X4 < 0.515]
        [X1 < 114.806]
         [X1 < 110.128]
          [0.0]
          [0.0]
         [X1 < 114.806]
          [0.0]
          [0.0]
        [1.0]
       [X4 < 0.333]
        [X1 < 210.289]
         [X1 < 128.703]
          [0.0]
          [X1 < 128.703]
           [0.0]
           [0.0]
         [1.0]
        [X1 < 224.684]
         [X1 < 111.605]
          [1.0]
          [1.0]
         [1.0]
      [X3 < 2.490]
       [X9 < 30.455]
        [X1 < 68.434]
         [0.0]
         [0.0]
        [X1 < 99.642]
         [X5 < 0.182]
          [0.0]
          [X10 < 450.402]
           [1.0]
           [0.0]
         [X1 < 171.209]
```

```
          [X2 < 11.911]
           [1.0]
           [0.0]
          [1.0]
       [X4 < 0.323]
        [X4 < 0.321]
         [X7 < 58.596]
          [X8 < 15.326]
           [1.0]
           [1.0]
          [X10 < 94.769]
           [0.0]
           [1.0]
         [X1 < 62.326]
          [0.0]
          [0.0]
        [X1 < 272.063]
         [X2 < 31.633]
          [X3 < 2.591]
           [1.0]
           [1.0]
          [X6 < -28.851]
           [0.0]
           [1.0]
         [0.0]
     [X3 < 4.798]
      [X9 < 31.642]
       [X6 < -34.984]
        [X7 < 88.192]
         [X1 < 107.104]
          [X1 < 62.206]
           [1.0]
           [1.0]
          [X1 < 107.104]
           [1.0]
           [1.0]
         [X7 < 89.837]
          [X1 < 80.709]
           [0.0]
           [0.0]
          [X1 < 121.937]
           [1.0]
           [1.0]
        [X1 < 64.525]
         [X2 < 22.314]
          [1.0]
          [X1 < 63.709]
           [0.0]
           [0.0]
         [X8 < -10.229]
          [X1 < 115.082]
           [0.0]
           [1.0]
          [X7 < 54.170]
           [1.0]
           [1.0]
       [X10 < 139.354]
```

```
    [X1 < 61.654]
     [X3 < 3.079]
      [X1 < 58.126]
       [1.0]
       [1.0]
      [X6 < 55.352]
       [0.0]
       [1.0]
     [X10 < 139.291]
      [X7 < 94.129]
       [1.0]
       [1.0]
      [0.0]
    [X3 < 3.065]
     [X3 < 3.063]
      [X6 < 219.896]
       [1.0]
       [1.0]
      [0.0]
     [X5 < 0.019]
      [X5 < 0.019]
       [1.0]
       [0.0]
      [X2 < 34.216]
       [1.0]
       [1.0]
  [X1 < 166.073]
   [X1 < 150.978]
    [X1 < 129.785]
     [0.0]
     [0.0]
    [0.0]
   [X4 < 0.016]
    [X1 < 235.552]
     [0.0]
     [1.0]
    [X1 < 216.824]
     [X1 < 190.008]
      [X1 < 166.073]
       [1.0]
       [1.0]
      [1.0]
     [X1 < 216.824]
      [1.0]
      [1.0]
```

Task 2: [2] Implement a 10-fold cross-validation to evaluate the model.

EXPLAINATION & STEPS FOR 10 FOLD CROSS-VALIDATION ON MODEL AS SHOWN BELOW:

[1] We first call the Load csv() function to load the magic04dataset after converting all data string to index in the document.

[2] We then use the eval_algorithm() which then implements the 10 fold cross-validation. Meaning, firstly the data will be split into 10 folds equally, followed by measuring the accuracy by using the calc_accuracy_metric() function.

In [27]:
```python
#  from random import seed  -- ###
from random import randrange
from csv import reader

### -- Load a CSV file -- ###
def load_csv(filename):
    file = open(filename, "rt")
    lines = reader(file)
    dataset = list(lines)
    return dataset

### -- Convert string column to float -- ###
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

### -- Function to split dataset into k folds -- ###
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

### -- Calc accuracy percentage % -- ###
def calc_accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

### -- Evaluate an algorithm using a cross validation split -- ###
def eval_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = calc_accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores
```

```
### -- Split a dataset based on an attribute and an attribute value  -- ###
def testing_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right

### -- Calculate the Gini index for a split dataset  -- ###
def gini_index(groups, classes):
    n_instances = float(sum([len(group) for group in groups]))
    gini = 0.0
    for group in groups:
        size = float(len(group))
        if size == 0:
            continue
        score = 0.0
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        gini += (1.0 - score) * (size / n_instances)
    return gini

### -- Select the best split point for a dataset -- ###
def getData_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = testing_split(index, row[index], dataset)
            gini = gini_index(groups, class_values)
            if gini < b_score:
                b_index, b_value, b_score, b_groups = index, row[index], gini,
groups
    return {'index':b_index, 'value':b_value, 'groups':b_groups}

### -- Create a terminal node value  -- ###
def to_terminal(group):
    outcomes = [row[-1] for row in group]
    return max(set(outcomes), key=outcomes.count)

### -- Create child splits for a node or make terminal  -- ###
def split(node, max_depth, min_size, depth):
    left, right = node['groups']
    del(node['groups'])
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left), to_terminal(right)
        return
    if len(left) <= min_size:
        node['left'] = to_terminal(left)
    else:
        node['left'] = getData_split(left)
```

```
        split(node['left'], max_depth, min_size, depth+1)
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    else:
        node['right'] = getData_split(right)
        split(node['right'], max_depth, min_size, depth+1)

### --  Build Decision Tree  -- ###
def build_tree(train, max_depth, min_size):
    root = getData_split(train)
    split(root, max_depth, min_size, 1)
    return root

# Make a prediction with a decision tree  -- ###
def predict(node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return predict(node['right'], row)
        else:
            return node['right']

### -- Classification & Regression Algorithm -- ###
def decision_tree(train, test, max_depth, min_size):
    tree = build_tree(train, max_depth, min_size)
    predictions = list()
    for row in test:
        prediction = predict(tree, row)
        predictions.append(prediction)
    return(predictions)


seed(1)
filename = 'C:/Users/Isaac Yeo/Downloads/magic04.data'
dataset = load_csv(filename)
### -- convert string attributes to integers  -- ###
for i in range(len(dataset[0])):
    str_column_to_float(dataset, i)
### -- evaluation algorithm  -- ###
n_folds = 10
max_depth = 5
min_size = 10
scores = eval_algorithm(dataset, decision_tree, n_folds, max_depth, min_size)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Mean Accuracy: 97.445%
```

In [ ]: