## MACHINE LEARINING LAB ASSESSMENT – I 15BCE0082 VOLETI RAVI

## CODE:

```
#Split a dataset based on an attribute and an attribute value
def test_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):
       # 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))
               # avoid divide by zero
               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
               # weight 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 get_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 = test_split(index, row[index], dataset)
                       gini = gini_index(groups, class_values)
                      print('X\%d < \%.3f Gini=\%.3f'\% ((index+1), row[index], gini))
                       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'])
       # check for a no split
       if not left or not right:
               node['left'] = node['right'] = to_terminal(left + right)
               return
       # check for max depth
       if depth >= max_depth:
               node['left'], node['right'] = to_terminal(left), to_terminal(right)
       # process left child
       if len(left) <= min_size:</pre>
               node['left'] = to_terminal(left)
       else:
               node['left'] = get split(left)
               split(node['left'], max_depth, min_size, depth+1)
       # process right child
       if len(right) <= min size:
               node['right'] = to_terminal(right)
       else:
               node['right'] = get_split(right)
               split(node['right'], max_depth, min_size, depth+1)
# Build a decision tree
def build_tree(train, max_depth, min_size):
       root = get_split(train)
       split(root, max depth, min size, 1)
       return root
"""# Print a decision tree
def print_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)))"""
dataset = [[2.771244718, 1.784783929, 0],
       [1.728571309, 1.169761413, 0],
       [3.678319846, 2.81281357, 0],
       [3.961043357,2.61995032,0],
       [2.999208922,2.209014212,0],
       [7.497545867, 3.162953546, 1],
```

```
[9.00220326,3.339047188,1],
       [7.444542326, 0.476683375, 1],
       [10.12493903,3.234550982,1],
       [6.642287351,3.319983761,1]]
tree = build_tree(dataset, 1, 1)
"""print tree(tree)"""
# Make a prediction with a decision tree
def predict(node, row):
       if row[node['index']] < node['value']:</pre>
               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']
# predict with a stump
stump = {'index': 0, 'right': 1, 'value': 6.642287351, 'left': 0}
for row in dataset:
       prediction = predict(stump, row)
       print('Expected=%d, Got=%d' % (row[-1], prediction))
```

## **OUTPUT:**

```
xelese@xelese-Lenovo-Y50-70: ~/Machine Learning/Decision_Tree
xelese@xelese-Lenovo-Y50-70:~/Machine Learning/Decision_Tree$ python DTree.py
X1 < 2.771 Gini=0.444
X1 < 1.729 Gini=0.500
X1 < 3.678 Gini=0.286
X1 < 3.961 Gini=0.167
X1 < 2.999 Gini=0.375
      7.498 Gini=0.286
     9.002 Gini=0.375
     7.445 Gini=0.167
X1 < 10.125 Gini=0.44
X1 < 6.642 Gini=0.000
X2 < 1.785 Gini=0.500
X2 < 1.170 Gini=0.444
   < 10.125 Gini=0.444
   < 6.642 Gini=0.000
      2.813 Gini=0.320
     2.620 Gini=0.417
     2.209 Gini=0.476
X2 < 3.163 Gini=0.167
   < 3.339 Gini=0.444
   < 0.477 Gini=0.500
   < 3.235 Gini=0.286
X2 < 3.320 Gini=0.375
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
xelese@xelese-Lenovo-Y50-70:~/Machine Learning/Decision_Tree$
```