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)

return

# process left child

if len(left) <= min\_size:

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']:

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

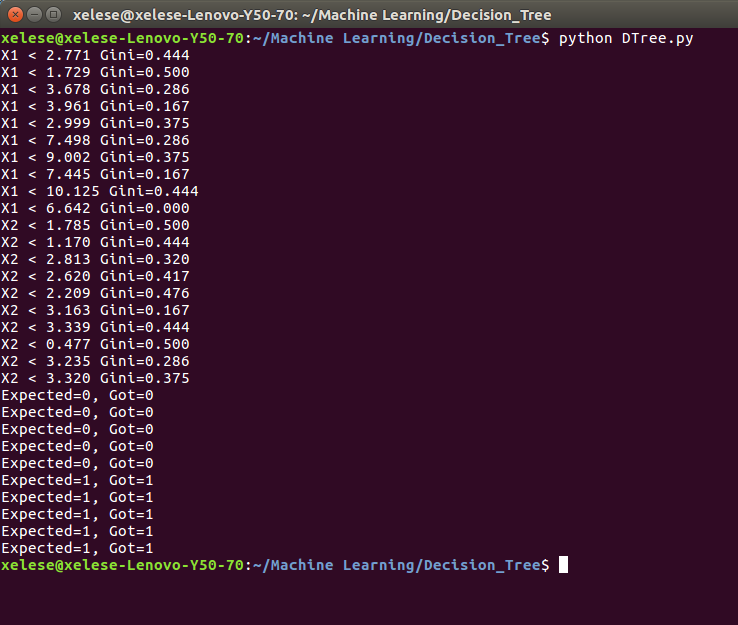
stump = {'index': 0, 'right': 1, 'value': 6.642287351, 'left': 0}

for row in dataset:

prediction = predict(stump, row)

print('Expected=%d, Got=%d' % (row[-1], prediction))

**OUTPUT:**

****