In [32]:

```
# Problem 1

# Method 1

import pandas as pd

balance_data = pd.read_csv('C://Users/a_shy/Desktop/TUM/Sem 1/Machine Learning/Homework/HW1
dataset = balance_data.values[:, 0:4]

# 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
    print('Split: [X%d < %.3f]' % ((b_index +1), b_value))
    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)))

tree = build_tree(dataset, 3, 1)
print_tree(tree)

#Output :
X1 < 5.500 Gini=0.381
X1 < 7.400 Gini=0.489
X1 < 5.900 Gini=0.445
X1 < 9.900 Gini=0.610
X1 < 6.900 Gini=0.536
X1 < 6.800 Gini=0.533
X1 < 4.100 Gini=0.387
X1 < 1.300 Gini=0.522
X1 < 4.500 Gini=0.296
X1 < 0.500 Gini=0.619
X1 < 5.900 Gini=0.445
X1 < 9.300 Gini=0.554
X1 < 1.000 Gini=0.574
X1 < 0.400 Gini=0.658
X1 < 2.700 Gini=0.461
X2 < 0.500 Gini=0.621
X2 < 1.100 Gini=0.610
X2 < 0.200 Gini=0.627
```

```
X2 < 0.100 Gini=0.655

X2 < -0.100 Gini=0.652
X2 < -0.300 Gini=0.619
X2 < 0.300 Gini=0.639
X2 < -0.200 Gini=0.631
X2 < 0.400 Gini=0.589
X2 < 0.000 Gini=0.652
X2 < -0.100 Gini=0.652
X2 < -0.200 Gini=0.631
X2 < 0.100 Gini=0.655
X2 < 0.100 Gini=0.655
X2 < -0.500 Gini=0.658
X3 < 4.500 Gini=0.589
X3 < 3.600 Gini=0.655
X3 < 3.400 Gini=0.636
X3 < 0.800 Gini=0.600
X3 < 0.600 Gini=0.658
X3 < 5.100 Gini=0.631
X3 < 5.100 Gini=0.631
X3 < 1.800 Gini=0.621
X3 < 2.000 Gini=0.656
X3 < 2.300 Gini=0.639
X3 < 4.400 Gini=0.627
X3 < 3.200 Gini=0.644
X3 < 2.800 Gini=0.653
X3 < 4.300 Gini=0.640
X3 < 4.200 Gini=0.630
Split: [X1 < 4.500]
X1 < 4.100 Gini=0.000
X1 < 1.300 Gini=0.000
X1 < 0.500 Gini=0.000
X1 < 1.000 Gini=0.000
X1 < 0.400 Gini=0.000
X1 < 2.700 Gini=0.000
X2 < 0.300 Gini=0.000
X2 < -0.200 Gini=0.000
X2 < 0.000 Gini=0.000
X2 < 0.100 Gini=0.000
X2 < 0.100 Gini=0.000
X2 < -0.500 Gini=0.000
X3 < 5.100 Gini=0.000
X3 < 1.800 Gini=0.000
X3 < 2.300 Gini=0.000
X3 < 2.800 Gini=0.000
X3 < 4.300 Gini=0.000
X3 < 4.200 Gini=0.000
Split: [X1 < 4.100]
X1 < 1.300 Gini=0.000
X1 < 0.500 Gini=0.000
X1 < 1.000 Gini=0.000
X1 < 0.400 Gini=0.000
X1 < 2.700 Gini=0.000
X2 < -0.200 Gini=0.000
X2 < 0.000 Gini=0.000
X2 < 0.100 Gini=0.000
X2 < 0.100 Gini=0.000
X2 < -0.500 Gini=0.000
X3 < 1.800 Gini=0.000
X3 < 2.300 Gini=0.000
X3 < 2.800 Gini=0.000
X3 < 4.300 Gini=0.000
```

```
X3 < 4.200 Gini=0.000

Split: [X1 < 1.300]
X1 < 5.500 Gini=0.444
X1 < 7.400 Gini=0.296
X1 < 5.900 Gini=0.492
X1 < 9.900 Gini=0.444
X1 < 6.900 Gini=0.433
X1 < 6.800 Gini=0.489
X1 < 4.500 Gini=0.494
X1 < 5.900 Gini=0.492
X1 < 9.300 Gini=0.381
X2 < 0.500 Gini=0.492
X2 < 1.100 Gini=0.444
X2 < 0.200 Gini=0.489
X2 < 0.100 Gini=0.489
X2 < -0.100 Gini=0.492
X2 < -0.300 Gini=0.494
X2 < 0.400 Gini=0.481
X2 < -0.100 Gini=0.492
X2 < -0.200 Gini=0.417
X3 < 4.500 Gini=0.317
X3 < 3.600 Gini=0.489
X3 < 3.400 Gini=0.433
X3 < 0.800 Gini=0.417
X3 < 0.600 Gini=0.494
X3 < 5.100 Gini=0.417
X3 < 2.000 Gini=0.492
X3 < 4.400 Gini=0.444
X3 < 3.200 Gini=0.481
Split: [X1 < 7.400]
X1 < 5.500 Gini=0.267
X1 < 5.900 Gini=0.417
X1 < 6.900 Gini=0.400
X1 < 6.800 Gini=0.333
X1 < 4.500 Gini=0.444
X1 < 5.900 Gini=0.417
X2 < 0.500 Gini=0.400
X2 < 0.200 Gini=0.444
X2 < -0.100 Gini=0.400
X2 < -0.300 Gini=0.444
X2 < 0.400 Gini=0.417
X2 < -0.100 Gini=0.400
X3 < 4.500 Gini=0.333
X3 < 3.400 Gini=0.417
X3 < 0.600 Gini=0.444
X3 < 5.100 Gini=0.400
X3 < 2.000 Gini=0.400
X3 < 4.400 Gini=0.444
Split: [X1 < 5.500]
X1 < 7.400 Gini=0.000
X1 < 9.900 Gini=0.000
X1 < 9.300 Gini=0.000
X2 < 1.100 Gini=0.000
X2 < 0.100 Gini=0.000
X2 < -0.200 Gini=0.000
X3 < 3.600 Gini=0.000
X3 < 0.800 Gini=0.000
X3 < 3.200 Gini=0.000
Split: [X1 < 7.400]
[X1 < 4.500]
 [X1 < 4.100]
```

```
  [X1 < 1.300]
   [1.0]
   [1.0]
  [1.0]
 [X1 < 7.400]
  [X1 < 5.500]
   [0.0]
   [2.0]
  [X1 < 7.400]
   [0.0]
   [0.0]
```

```
# Method 2

import numpy as np
import pandas as pd

from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree

balance_data = pd.read_csv('C://Users/a_shy/Desktop/TUM/Sem 1/Machine Learning/Homework/HW1

X = balance_data.values[:, 0:3]  # Separate attribute vectors
Y = balance_data.values[:,3]  # Separate target vectors

clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3, min_s
clf_gini.fit(X,Y)

# Save tree as dot file
with open("tree1.dot", 'w') as f:
  f = tree.export_graphviz(clf_gini, out_file=f)


# Tree plot

digraph Tree {
node [shape=box] ;
0 [label="X[0] <= 4.3\ngini = 0.6578\nsamples = 15\nvalue = [5, 6, 4]"] ;
1 [label="gini = 0.0\nsamples = 6\nvalue = [0, 6, 0]"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="X[0] <= 7.15\ngini = 0.4938\nsamples = 9\nvalue = [5, 0, 4]"] ;
0 -> 2 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
3 [label="X[0] <= 5.0\ngini = 0.4444\nsamples = 6\nvalue = [2, 0, 4]"] ;
2 -> 3 ;
4 [label="gini = 0.0\nsamples = 1\nvalue = [1, 0, 0]"] ;
3 -> 4 ;
5 [label="gini = 0.32\nsamples = 5\nvalue = [1, 0, 4]"] ;
3 -> 5 ;
6 [label="gini = 0.0\nsamples = 3\nvalue = [3, 0, 0]"] ;
2 -> 6 ;
}


# Problem 2

# xa = (4.1, -0:1, 2.2)T lies in region 1 of above tree which has 6 samples
# so p(c = ya | xa, T) = 6/15 = 0.4
```

```
# xb = (6.1, 0.4, 1.3)T  lies is region 4 of above tree which has 1 sample
# so p(c = yb | xb, T) = 1/15 = 0.0667
```

In [ ]:

In [ ]:

In [ ]: