In [2]:
```
import numpy as np
import math
import sys
import os
```

In [3]:
```
def readfile(filename):
    X = []
    Y = []
    for lines in open(filename).readlines():
        temp = lines.strip().split()
        x = []
        for feature in temp[:-1]:
            x.append(float(feature))
        X.append(x)
        Y.append(int(float(temp[-1])))
    return np.asarray(X), np.asarray(Y)
```

In [4]:
```
def Gini_impurity(x, y, feature, theta, num_l, num_r):
    N = np.shape(x)[0]
    lp, ln, rp, rn = 0, 0, 0, 0
    for i in range(N):
        if x[:, feature][i] > theta:
            if y[i]==1:
                rp += 1
            rn = num_r-rp
        else:
            if y[i]==1:
                lp += 1
            ln = num_l-lp
    l_gini = 1-((lp/num_l)**2)-((ln/num_l)**2)
    r_gini = 1-((rp/num_r)**2)-((rn/num_r)**2)
    impurity = num_l*l_gini + num_r*r_gini
    return impurity

def select_theta(x, y, feature):
    v = sorted(set(x[:, feature]))
    values = np.array(v)
    N = np.shape(x)[0]
    min_impurity, best_theta = 10000000, None
    left = 0
    right = len(values)
    for i in range(len(values)-1):
        theta = (values[i]+values[i+1]) / 2
        left += 1
        right -= 1
        impurity = Gini_impurity(x, y, feature, theta, left, right)
        if impurity<min_impurity:
            min_impurity, best_theta = impurity, theta
    return min_impurity, best_theta

def select_feature(x, y):
    min_impurity, best_theta, best_feature = 10000000, None, None
    for i in range(10):
        impurity, theta = select_theta(x, y, i)
        if impurity<min_impurity:
            min_impurity, best_theta, best_feature = impurity, thet
a, i
    return best_theta, best_feature
```

In [5]:
```
class Node:
    def __init__(self, theta, feature):
        self.left = None
        self.right = None
        self.theta = theta
        self.feature = feature
        self.is_leaf = False
        self.predict = None
def binary_tree(x, y):
    y_list = y.tolist()
    p = y_list.count(1)
    n = y_list.count(-1)
    ## terminate => return g(t)
    if (p==0) or (n==0):
        leaf = Node(None,None)
        leaf.is_leaf = True
        if p>0: leaf.predict =  1
        else:   leaf.predict = -1
        return leaf
    elif (x!=x[0]).sum()==0:
        leaf = Node(None,None)
        leaf.is_leaf = True
        leaf.predict = -1
        return leaf
    ## no terminate
    else:
        ## learn branching criteria
        theta, feature = select_feature(x, y)
        ## split D to 2 parts = {X[:, feature]<=theta}{X[:, feature
]>theta}
        x1, y1, x2, y2 = [], [], [], []
        N = np.shape(x)[0]
        split = np.where(x[:, feature] > theta, 1, -1)
        for i in range(N):
            if split[i]==-1:
                x1.append(x[i])
                y1.append(y[i])
            elif split[i]==1:
                x2.append(x[i])
                y2.append(y[i])
        ## build two sub-tree
        if(len(y1)==0)or(len(y2)==0):
            split_list = split.tolist()

        tree = Node(theta, feature)
        tree.left  = binary_tree(np.asarray(x1), np.asarray(y1))
        tree.right = binary_tree(np.asarray(x2), np.asarray(y2))
        ## return tree
        return tree
```

In [6]:
```
def get_predict_label(node, xn):
    if node.is_leaf:
        return node.predict
    if xn[node.feature] <= node.theta:
        return get_predict_label(node.left, xn)
    elif xn[node.feature] > node.theta:
        return get_predict_label(node.right, xn)
def predict(x, y, root):
    N, correct = np.shape(x)[0], 0
    pre = []
    for i in range(N):
        predict_label = get_predict_label(root, x[i])
        pre.append(predict_label)
        if predict_label==y[i]:
            correct += 1
    return pre, round(correct/N, 3)
```

In [7]:
```
X, Y = readfile("hw6_train.dat")
Xt, Yt = readfile("hw6_test.dat")
```

In [9]:
```
## 14.
root = binary_tree(X, Y)
Train_Pre, Train_Acc = predict(X, Y, root)
Test_Pre, Test_Acc = predict(Xt, Yt, root)
print("14. Eout: ", 1-Test_Acc)
```

```
14. Eout:  0.16600000000000004
```

In [12]:
```
T = sys.argv[1]
idx = np.random.randint(1000, size=500)
Xb = X[idx, :]
Yb = Y[idx]
root = binary_tree(Xb, Yb)
np.save('Choose_points-'+str(T), idx)
np.save('Train_Pre-'+str(T), Train_Pre)
np.save('Train_Acc-'+str(T), Train_Acc)
np.save('Test_Pre-'+str(T), Test_Pre)
np.save('Test_Acc-'+str(T), Test_Acc)
```

In [15]:
```
## 15
test_acc = []
for i in range(4):
    for j in range(400):
        if os.path.isfile(str(i) + "/Test_Acc-" + str(j) + ".npy"):
            test_acc.append(float(np.load(str(i) + "/Test_Acc-" + s
tr(j) + ".npy")))
print('15. Average Eout:', 1-np.mean(test_acc))
```

```
15. Average Eout: 0.24629232039636662
```

In [18]:
```
## 16 17
G_train = np.zeros(1000)
G_test = np.zeros(1000)
for i in range(4):
    for j in range(400):
        if  os.path.isfile(str(i) + "/Train_Pre-" + str(j) + ".npy"
):
            Train_Pre  = np.load(str(i) + "/Train_Pre-" + str(j) +
".npy")
            G_train += Train_Pre
        if  os.path.isfile(str(i) + "/Test_Pre-" + str(j) + ".npy")
:
            Test_Pre  = np.load(str(i) + "/Test_Pre-" + str(j) + ".
npy")
            G_test += Test_Pre
G_Ein, G_Eout = 0, 0
for i in range(1000):
    if np.sign(G_train[i])!=Y[i]:
        G_Ein += 1
    if np.sign(G_test[i])!=Yt[i]:
        G_Eout += 1
print('16. G_Ein:', G_Ein/1000)
print('17. G_Eout:', G_Eout/1000)
```

```
16. G_Ein: 0.015
17. G_Eout: 0.171
```

In [27]:
```
## 18
OOB = np.zeros(1000)
for i in range(4):
    for j in range(400):
        if  os.path.isfile(str(i) + "/Choose_points-" + str(j) + ".
npy"):
            Choose_points = np.load(str(i) + "/Choose_points-" + st
r(j) + ".npy")
        if  os.path.isfile(str(i) + "/Train_Pre-" + str(j) + ".npy"
):
            Train_Pre = np.load(str(i) + "/Train_Pre-" + str(j) + "
.npy")
        for k in range(1000):
            if k in Choose_points:
                continue
            else:
                OOB[k] += Train_Pre[k]
Eoob = 0
for i in range(1000):
    if np.sign(OOB[i])!=Y[i]:
        Eoob += 1
print('18. Eoob:', Eoob/1000)
```

```
18. Eoob: 0.078
```