```
import numpy as np
import pandas as pd
from scipy import io
import random
import sys

class DecisionTree(object):
        class_label = 'label'
        def __init__(self, root=None):
                self.root = root

        def train(self, data, depth, random_forest=False):
                threshold, split_feature = self.segmenter(data, random_forest)
                labels = data[[DecisionTree.class_label]]
                if threshold > 0 and depth > 0:
                        root = Node()
                        root.split_rule = (split_feature, threshold)
                        left_data = data[data[split_feature] < threshold].reset_inde
x(drop=True)
                        left_labels = left_data[[DecisionTree.class_label]]
                        if pd.DataFrame.sum(left_labels[DecisionTree.class_label]) =
= 0:
                                root.left = LeafNode(0)
                        elif pd.DataFrame.sum(left_labels[DecisionTree.class_label])
 == len(left_labels):
                                root.left = LeafNode(1)
                        else:
                                root.left = self.train(left_data, depth - 1)

                        right_data = data[data[split_feature] >= threshold].reset_in
dex(drop=True)
                        right_labels = right_data[[DecisionTree.class_label]]
                        if pd.DataFrame.sum(right_labels[DecisionTree.class_label])
== 0:
                                root.right = LeafNode(0)
                        elif pd.DataFrame.sum(right_labels[DecisionTree.class_label]
) == len(right_labels):
                                root.right = LeafNode(1)
                        else:
                                root.right = self.train(right_data, depth - 1)
                else:
                        if pd.DataFrame.sum(labels[DecisionTree.class_label]) / len(
data) < 0.5:
                                root = LeafNode(0)
                        else:
                                root = LeafNode(1)

                print("DEPTH: ", depth)
                if root.type() == "Node":
                        print("THRESH: ", threshold)
                        print("SF: ", split_feature)
                elif root.type() == "LeafNode":
                        print("LABEL: ", root.label)
                print(" ")
                return root

        def predict(self, data):
                predicted_labels = []
                data_columns = list(data)
                temp = data.values[:10]
                for d in data.values:
                        curr_node = self.root
                        label_to_val = dict()
                        for i in range(len(data_columns)):
                                label_to_val[data_columns[i]] = d[i]
                        while curr_node.type() == "Node":
                                split_feature, threshold = curr_node.split_rule
                                dict_val = label_to_val[split_feature]
                                if (dict_val < threshold):
                                        curr_node = curr_node.left
                                else:
                                        curr_node = curr_node.right
                        predicted_labels.append(curr_node.label)
```

```
                return predicted_labels

        def impurity(self, left_label_hist, right_label_hist):
                left = sum(left_label_hist)
                right = sum(right_label_hist)

                if left != 0 and left_label_hist[0] != 0 and left_label_hist[1] != 0
:
                        left_0_label = left_label_hist[0] / float(left)
                        left_1_label = left_label_hist[1] / float(left)
                        P_left = -((left_0_label * np.log2(left_0_label)) + (left_1_
label * np.log2(left_1_label)))
                else:
                        P_left = 0

                if right != 0 and right_label_hist[0] != 0 and right_label_hist[1] !
= 0:
                        right_0_label = right_label_hist[0] / float(right)
                        right_1_label = right_label_hist[1] / float(right)
                        P_right = -((right_0_label * np.log2(right_0_label)) + (righ
t_1_label * np.log2(right_1_label)))
                else:
                        P_right = 0

                return ((P_left*left) + (P_right*right)) / (left + right)

        def segmenter(self, data, random_forest=False):
                data_length, features = np.shape(data)
                feature_list = list(data)
                feature_list.remove(DecisionTree.class_label)
                split_feature = -1
                best_threshold = -1
                best_impurity = 1

                labels = data[[DecisionTree.class_label]]
                root_1_labels = pd.DataFrame.sum(labels[DecisionTree.class_label])
                root_0_labels = data_length - root_1_labels

                feature_map = dict()
                for feat in list(data):
                        feature_map[feat] = data.columns.get_loc(feat)

                if random_forest == True:
                        print("rf segmenter")
                        delete_features_list = []
                        num_features = len(feature_list)
                        feature_list = random.sample(feature_list, int(np.sqrt(num_f
eatures)))
                        rf_map = dict()
                        for feat in feature_list:
                                rf_map[feat] = data.columns.get_loc(feat)
                        rf_map[DecisionTree.class_label] = feature_map[DecisionTree.
class_label]
                        feature_map = rf_map

                for i in range(len(feature_list)):
                        data = data.sort(feature_list[i])
                        #unique_thresholds = pd.Series.unique(data[feature_list[i]])
                        label_index = 0
                        count_1 = 0
                        count_0 = 0
                        curr_threshold = None
                        for d in data.values:
                                d_index = feature_map[feature_list[i]]
                                if d[d_index] != curr_threshold:
                                        curr_threshold = d[d_index]
                                        left_1_labels = count_1
                                        left_0_labels = count_0
                                        right_1_labels = root_1_labels - left_1_labe
ls
                                        right_0_labels = root_0_labels - left_0_labe
ls
                                        left_label_hist = (left_0_labels, left_1_lab
```

```
els)
                                        right_label_hist = (right_0_labels, right_1_
labels)
                                        curr_impurity = self.impurity(left_label_his
t, right_label_hist)
                                        if curr_impurity < best_impurity:
                                                best_impurity = curr_impurity
                                                split_feature = feature_list[i]
                                                best_threshold = curr_threshold
                                if d[feature_map[DecisionTree.class_label]] == 1:
                                        count_1 += 1
                                else:
                                        count_0 += 1
                                label_index += 1
                return (best_threshold, split_feature)

class Node(object):
        def __init__(self, left=None, right=None, split_rule=None):
                self.left = left
                self.right = right
                self.split_rule = split_rule

        def type(self):
                return "Node"

class LeafNode(object):
        def __init__(self, label):
                self.label = label

        def type(self):
                return "LeafNode"
```