```
"""
A FAKE Random Tree Learner.  (c) 2016 Tucker Balch
This is just a linear regression learner implemented named RTLearner so
it is used as a template or placeholder for use by testbest4.  You should
replace this code with your own RTLearner.
"""

import numpy as np
import pandas as pd
import random

#!!!!DEBUG Statement, set to FALSE when submitting
debug=False
def log(s):
    if debug:
        print s

class RTLearner(object):
    def __init__(self, verbose=False, leaf_size=1, col=None, value=None,
results=None,fb=None,tb=None):
    #!!! initialize tree just wit max leaf_size

        self.leaf_size=leaf_size
        self.tree=None
        self.temp_tree=None

        self.col = col  # col that splits on
        self.value = value  # split value
        self.results = results #final resulst
        self.tb = tb
        self.fb = fb

    # Note: the core ideas and some lines for the code below were borrowed from
the following website:
    # http://www.patricklamle.com/Tutorials/Decision%20tree
%20python/tuto_decision%20tree.html
    # I did have to adapt the code substantially to work with non-
string/classification (ie continuous) type DTs
    # I also had to modify it to work in python 2.7 (it was in python 3.0)
    def build_tree(self, data):

        if len(data) <= self.leaf_size or len(np.unique(data.iloc[:, -1])) <= 1:
            return RTLearner(results=np.mean(data.iloc[:, -1]))

        # randomly pick a random feature i of data to split on
        #id those columns that have split potential
        col_list=range(0,data.shape[1]-1)
        random.shuffle(col_list)

        split_ix=next(x for x in col_list if len(np.unique(data.iloc[:,x])) > 1)
        random_vals = np.random.choice(np.unique(data.iloc[:, split_ix]), 2,
replace=False)

        # if only one unique value use that
        #elif len(np.unique(data.iloc[:, i])) == 1:
        #    random_vals = data.iloc[0, i]
        # get mean of those values
        split_val = np.mean(random_vals)
        if len(data) > self.leaf_size and len(np.unique(data.iloc[:, -1])) > 1:

            fb = self.build_tree(data=data[data.iloc[:, split_ix] <= split_val])
            tb = self.build_tree(data=data[data.iloc[:, split_ix] > split_val])
            return RTLearner(col=split_ix, value=split_val, fb=fb, tb=tb)
        else:
```

```
            log("YOU SHOULD NOT REACH HERE")

    # the following code is helpful for visualizing the tree
    def printtree(self,indent=''):
        # Is this a leaf node?
        if self.tree.results != None:
            print(str(self.tree.results))
        else:
            #print the column ix of interest for splitting, and the value that
was split on
            print(str(self.tree.col) + ':' + str(self.tree.value) + '? ')
            # Print the branches for the next trees
            print(indent + 'T->'),
            self.tree=self.tree.tb
            self.printtree(indent + '  ')
            print(indent + 'F->'),
            self.tree=self.tree.fb
            self.printtree(indent + '  ')

    #!!!bring in the data
    def addEvidence(self, trainX, trainY):
        #join the data so can track the predictions as we manipulate
        data=np.concatenate((trainX,trainY[:, None]),axis=1)
        data=pd.DataFrame(data)
        self.tree=self.build_tree(data)

    """
    @summary: Add training data to learner
    @param dataX: X values of data to add
    @param dataY: the Y training values


    # slap on 1s column so linear regression finds a constant term
    newdataX = np.ones([dataX.shape[0], dataX.shape[1] + 1])
    newdataX[:, 0:dataX.shape[1]] = dataX

    # build and save the model
    self.model_coefs, residuals, rank, s = np.linalg.lstsq(newdataX, dataY)


    """

    def classify_obs(self, obs):
        if self.temp_tree.results != None:
            return self.temp_tree.results
        obs_val = obs[self.temp_tree.col]  # take the obs value at the index in
the decision  tree
        if obs_val > self.temp_tree.value:
            self.temp_tree = self.temp_tree.tb  # go down the true branch
        else:
            self.temp_tree = self.temp_tree.fb
        return self.classify_obs(obs)
    #!!!use tree to classify new data points (first classify one datapoint then
apply to all)
    def query(self, points):
        # take the given tree and a new observation and return the value of
interest for that observation
        Y_pred=[]
        for row in points:
            self.temp_tree=self.tree
            Y_pred.append(self.classify_obs(row))
        return Y_pred
```

```
        """
        @summary: Estimate a set of test points given the model we built.
        @param points: should be a numpy array with each row corresponding to a
specific query.
        @returns the estimated values according to the saved model.

        # get the linear result
        ret_val = (self.model_coefs[:-1] * points).sum(axis=1) +
self.model_coefs[-1]
        # add some random noise
        ret_val = ret_val + 0.09 * np.random.normal(size=ret_val.shape[0])
        return ret_val
         """
    def author(self):
        return 'nbuckley7'  #  Georgia Tech username.

#if __name__ == "__main__":
#    print "get me a shrubbery"
```