```
import numpy as np

class DTLearner(object):

    def __init__(self, leaf_size = 1, verbose = False):
        self.leaf_size = leaf_size
        self.verbose = verbose
        self.leaf = None

    def author(self):
        return 'kkc3'

    def addEvidence(self, dataX, dataY):
        data_y = np.array([dataY])
        data = np.concatenate((dataX, data_y.T), axis=1)
        self.tree = self.build_tree(data)

        if self.verbose: print self.tree

    @staticmethod
    def is_Same(arr):
        for i in range(1,len(arr)):
            if arr[i] != arr[i-1]:
                return False
        return True

    @staticmethod
    def get_best_feature(X, Y):
        best_corr = -1.
        best_feat = -1
        for i in range(X.shape[1]):
            corr = abs(np.corrcoef(X[:,i], Y)[1,0])
            if corr > best_corr:
                best_corr = corr
                best_feat = i
        return best_feat

    def build_tree(self, data):
        X, Y = data[:,:-1], data[:,-1]
        if self.is_Same(Y) or X.shape[0] <= self.leaf_size:
            return np.array([self.leaf, np.mean(Y), np.nan, np.nan])
        else:
            b = self.get_best_feature(X, Y)
            splitval = np.median(X[:,b])
            split_left = data[data[:,b] <= splitval]
            split_right = data[data[:,b] > splitval]

            if split_left.shape[0] == X.shape[0] \
            or split_right.shape[0] == X.shape[0]:
                return np.array([self.leaf, np.mean(Y), np.nan, np.nan])

            left_tree = self.build_tree(split_left)
            right_tree = self.build_tree(split_right)

            if len(left_tree.shape) == 1:
                right_pointer = 1
            else:
                right_pointer = left_tree.shape[0]
```

```
            root = np.array([b, splitval, 1, right_pointer + 1])
            trees = np.vstack((left_tree, right_tree))
            return np.vstack((root, trees))

    def query(self, xtest):
        preds = []
        if xtest.ndim == 1: num_rows = 1
        else: num_rows = xtest.shape[0]

        for i in range(num_rows):
            curr = 0
            while self.tree[curr, 0] != None:
                feat = int(self.tree[curr,0])
                splitval = self.tree[curr,1]
                #attend to case where xtest has single dimension..can't double
index then
                if xtest.ndim == 1:
                    if xtest[feat] <= splitval: add = int(self.tree[curr,2])
                    else: add = int(self.tree[curr,3])
                else:
                    if xtest[i,feat] <= splitval:
                        add = int(self.tree[curr,2])
                    else:
                        add = int(self.tree[curr,3])
                curr = curr + add
                pred = self.tree[curr,1]
            preds.append(pred)
        return np.array(preds)
```