In [8]:

```
import numpy as np
import matplotlib.pyplot as plt
import random as rd


#класс ячейки дерева
class Cell:
    def __init__(self, data):
        self.data = data
        self.children = []
    def add_child(self, obj):
        self.children.append(obj)

def mse(info):
    mean = np.mean(info)
    result = 0
    for i in info:
        result += (i - mean) ** 2
    return result

#функция для критерия gini
def gini(info):
    answer = 0
    len_ = len(info)
    targets = set(info)
    probabilities = []
    for t in targets:
        _t = [i for i in info if i == t]
        probabilities.append(len(_t) * 1.0 / len_)
    for p in probabilities:
        answer = answer + p * (1 - p)
    return answer
```

In [9]:

```
class DecisionTree:
    def __init__(self, criterion = 'gini',  min_weight = 1, min_samples_split=3):
        self.cell = Cell(0)
        self.criterion = criterion
        self.min_weight = min_weight
        self.min_samples_split = min_samples_split

    def split(self, data, targets):
        #функция для выдачи ответа в листе
        def answer_in_leaf(info):
            if self.criterion == 'gini':
                return max(set(info), key = info.count)
            elif self.criterion == 'mse':
                return np.mean(info)

        #функция для определения разделения по данной фиче j и порогу threshold
        def accuracy(j, threshold):
            #j = feature
            func = None
            if self.criterion == 'gini':
                func = gini
            elif self.criterion == 'mse':
                func = mean_squared_error
            data1 = [data[i] for i in range(0, len(data)) if data[i][j] < threshold]
            targets1 = [targets[i] for i in range(0, len(data)) if data[i][j] < threshold]
            data2 = [data[i] for i in range(0, len(data)) if data[i][j] >= threshold]
            targets2 = [targets[i] for i in range(0, len(data)) if data[i][j] >= threshold]
            if(len(data1)  * len(data2) == 0):
                return -100
            answer = len(data1) * 1.0 / len(data) * func(targets1) + len(data2) * 1.0 / ler
            return answer


        #начинаем построение

        if len(data) < self.min_samples_split:
            return answer_in_leaf(targets)

        min_j = 0
        min_threshold = 0
        min_accuracy = -1
        for j in range(0, len(data[0])):
            values = [feature_value[j] for feature_value in data]
            thresholds = np.linspace(min(values), max(values), 100)
            for threshold_ in thresholds:
                if accuracy(j, threshold_) > 0:
                    min_j = j
                    min_threshold = threshold_
                    min_accuracy = accuracy(j, threshold_)
                    break

        if min_accuracy == -1:
            return answer_in_leaf(targets)

        for j in range(0, len(data[0])):
            values = [feature_value[j] for feature_value in data]
            thresholds = np.linspace(min(values), max(values), 100)
            for threshold_ in thresholds:
                if accuracy(j, threshold_) > min_accuracy and accuracy(j, threshold_) != -1
```

```
                    min_j = j

                    min_threshold = threshold_
                    min_accuracy = accuracy(j, threshold_)
        data1 = [data[i] for i in range(0, len(data)) if data[i][min_j] < min_threshold]
        targets1 = [targets[i] for i in range(0, len(data)) if data[i][min_j] < min_thresho
        data2 = [data[i] for i in range(0, len(data)) if data[i][min_j] >= min_threshold]
        targets2 = [targets[i] for i in range(0, len(data)) if data[i][min_j] >= min_thresh

        #переход на следующий уровень

        if (len(data1) >= self.min_weight and len(data2) >= self.min_weight):
            new_cell = Cell([min_j, min_threshold])
            _cell = self.split(data1, targets1)
            new_cell.add_child(_cell)
            _cell = self.split(data2, targets2)
            new_cell.add_child(_cell)
            return new_cell
        else:
            return answer_in_leaf(targets)

    def fit(self, data_, targets_):
        self.cell = self.split(data_, targets_)

    def predict(self, data_):
        targets_ = [0.0] * len(data_)
        count = 0
        for d in data_:
            cell_ = self.cell
            while type(cell_) != np.float64 and type(cell_) != np.float64:#спускаемся по де
                if d[cell_.data[0]] < cell_.data[1]:
                    cell_ = cell_.children[0]
                else:
                    cell_ = cell_.children[1]
            targets_[count] = cell_
            count +=1
        return targets_
```

In [10]:

```
#загружаем данные
from sklearn.datasets import load_boston
from sklearn import tree
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, classification_report
data = load_boston().data
target = load_boston().target

#используем наше дерево
train_data, test_data, train_target, test_target = train_test_split(data, target, test_size
tree_ = DecisionTree()
tree_.fit(train_data, train_target)
predictions = tree_.predict(test_data)
sigma = (abs(np.array(test_target) - np.array(predictions))).mean()
print sigma
```

```
4.76862745098
```

In [11]:

```
from sklearn import tree
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_absolute_error, classification_report
model = tree.DecisionTreeRegressor(max_depth = 15)
model.fit(train_data, train_target)
test_predictions = model.predict(test_data)
print mean_absolute_error(test_target, test_predictions)
```

```
3.03022875817
```

In [ ]:

```
#Видим, что качество довольно неплохое
```