10+CART(Regression)

2019 年 3 月 12 日

0.1 CART(Regression)

0.2 一、概念

回归树和分类树相似，区别在于划分变量的方法和叶节点预测结果的产生方式。

1. 划分方法：

- 分类树：**基尼系数**
- 回归树：**均方误差**

2. 预测方式

- 分类树：划分后的样本集的**目标变量的众数**
- 回归树：划分后的样本集的**目标变量的平均数**

假设输入空间有 $R_1, R_2, ..., R_M$ 个划分单元，回归树模型可以看作是针对每个划分单元上的固定输出值 $c_1, c_2, ..., c_M$

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

每个划分上的预测结果为：

$$\hat{c}_m = mean(y_i | x_i \in R_m)$$

选用第 j 个变量 x^{(j)} 和它取的值 s，作为切分变量和切分点，划定两个区域：

$$R_1(j,s) = \{x | x^{(j)} \leq s\} R_2(j,s) = \{x | x^{(j)} > s\}$$

寻找最优切分变量和切分点就是求解：

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

对固定的输入变量 j 可以找到最优切分点 s：

$$\hat{c}_1 = mean(y_i|x_i \in R_1)\hat{c}_2 = mean(y_i|x_i \in R_2)$$

遍历所有输入变量，找到最优分割变量的最优分割点分成两个区域，并对每个区域重复上述操作，直到满足停止条件为止。

0.3 二、划分方法：均方误差 vs 方差

均方误差和方差的区别在于常系数的不同，均方误差除以样本数（减一）就是方差。

以下给出几个例子对方差和均方误差进行比较：

1. 等差数列：

以数据的索引作为划分变量，假设数据是等差数列，也即数据是 y=0.1*x 直线上均匀的点。

In [2]:
```
import numpy as np

data = np.linspace(0.1,10, 100)
data
```

Out[2]:
```
array([ 0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,  1.1,
        1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,  2.2,
        2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  3.3,
        3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  4.4,
        4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,  5.5,
        5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,  6.6,
        6.7,  6.8,  6.9,  7. ,  7.1,  7.2,  7.3,  7.4,  7.5,  7.6,  7.7,
        7.8,  7.9,  8. ,  8.1,  8.2,  8.3,  8.4,  8.5,  8.6,  8.7,  8.8,
        8.9,  9. ,  9.1,  9.2,  9.3,  9.4,  9.5,  9.6,  9.7,  9.8,  9.9,
       10. ])
```

显然 numpy 的方差是近似方差，不是方差的无偏估计

In [13]:
```
np.var([0,2])
```

Out[13]:
```
1.0
```

In [39]:
```
def find(data, var=True):
    n = len(data)
    loss = float('inf')
    split = -1
    for i in range(1, n-1):
        a = data[0:i]
        b = data[i:n]
```

```
            if var:
                tmp = np.var(a) + np.var(b)
            else:
                tmp = len(a)*np.var(a) + len(b)*np.var(b)
            if tmp < loss:
                split = i
                loss = tmp
        return split, loss

    print(find(data, var=True))
    print(find(data, var=False))
```

```
(50, 4.165)
(50, 208.25)
```

显然，如果数据是严格的等差数列，则无论是方差还是均方误差，则划分点都是中位数

2. 数据是不对称的，增减性在转折点前后相反的折线

```
In [25]: data = np.append(data, np.linspace(10, 1, 10))
         print(data)
```

```
[ 0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4
  1.5  1.6  1.7  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8
  2.9  3.   3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.   4.1  4.2
  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.   5.1  5.2  5.3  5.4  5.5  5.6
  5.7  5.8  5.9  6.   6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9  7.
  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.   8.1  8.2  8.3  8.4
  8.5  8.6  8.7  8.8  8.9  9.   9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8
  9.9 10.  10.   9.   8.   7.   6.   5.   4.   3.   2.   1. ]
```

```
In [26]: print(find(data, var=True))
         print(find(data, var=False))
```

```
(37, 5.355548883467819)
(47, 323.36984126984123)
```

```
In [30]: print(data[37])
         print(np.mean(data[0: 37]))
         print(np.mean(data[37: 110]))
```

```
3.8000000000000003
1.9
6.708219178082191
```

```
In [31]: pritn(data[47])
         print(np.mean(data[0: 47]))
         print(np.mean(data[47: 110]))
```

```
2.4
7.098412698412699
```

可见，是方差的情况下，分离点更倾向于偏离转折点，是均方误差的情况下分离点更接近转折点

3. 数据是对称的，增减性在转折点前后相反的折线

```
In [32]: data = np.linspace(0.1,10, 100)
         data = np.append(data, np.linspace(10,0.1, 100))
         print(data)
```

```
[ 0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4
  1.5  1.6  1.7  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8
  2.9  3.   3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.   4.1  4.2
  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.   5.1  5.2  5.3  5.4  5.5  5.6
  5.7  5.8  5.9  6.   6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9  7.
  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.   8.1  8.2  8.3  8.4
  8.5  8.6  8.7  8.8  8.9  9.   9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8
  9.9 10.  10.   9.9  9.8  9.7  9.6  9.5  9.4  9.3  9.2  9.1  9.   8.9
  8.8  8.7  8.6  8.5  8.4  8.3  8.2  8.1  8.   7.9  7.8  7.7  7.6  7.5
  7.4  7.3  7.2  7.1  7.   6.9  6.8  6.7  6.6  6.5  6.4  6.3  6.2  6.1
  6.   5.9  5.8  5.7  5.6  5.5  5.4  5.3  5.2  5.1  5.   4.9  4.8  4.7
  4.6  4.5  4.4  4.3  4.2  4.1  4.   3.9  3.8  3.7  3.6  3.5  3.4  3.3
  3.2  3.1  3.   2.9  2.8  2.7  2.6  2.5  2.4  2.3  2.2  2.1  2.   1.9
  1.8  1.7  1.6  1.5  1.4  1.3  1.2  1.1  1.   0.9  0.8  0.7  0.6  0.5
  0.4  0.3  0.2  0.1]
```

```
In [33]: print(find(data, var=True))
         print(find(data, var=False))
```

```
(182, 7.572185927625489)
(38, 1215.6604938271605)
```

```
In [34]: print(data[182])
         print(data[38])
```

```
1.7999999999999999
3.9000000000000004
```

数据是对称的情况下分割点应该也是对称等价的，所以方差分割点 182 应该和 18 是等价的，只是因为计算机精度的问题造成结果是 182。

可以发现在方差的条件下分割点对比均方误差条件仍然是远离转折点。

4. 数据是单调前后斜率不同的折线

```
In [44]: data = np.linspace(1,100, 100)
         data = np.append(data, np.linspace(100.1,110, 100))
         print(data)
```

```
[  1.    2.    3.    4.    5.    6.    7.    8.    9.   10.   11.   12.
  13.   14.   15.   16.   17.   18.   19.   20.   21.   22.   23.   24.
  25.   26.   27.   28.   29.   30.   31.   32.   33.   34.   35.   36.
  37.   38.   39.   40.   41.   42.   43.   44.   45.   46.   47.   48.
  49.   50.   51.   52.   53.   54.   55.   56.   57.   58.   59.   60.
  61.   62.   63.   64.   65.   66.   67.   68.   69.   70.   71.   72.
  73.   74.   75.   76.   77.   78.   79.   80.   81.   82.   83.   84.
  85.   86.   87.   88.   89.   90.   91.   92.   93.   94.   95.   96.
  97.   98.   99.  100.  100.1 100.2 100.3 100.4 100.5 100.6 100.7 100.8
 100.9 101.  101.1 101.2 101.3 101.4 101.5 101.6 101.7 101.8 101.9 102.
 102.1 102.2 102.3 102.4 102.5 102.6 102.7 102.8 102.9 103.  103.1 103.2
 103.3 103.4 103.5 103.6 103.7 103.8 103.9 104.  104.1 104.2 104.3 104.4
 104.5 104.6 104.7 104.8 104.9 105.  105.1 105.2 105.3 105.4 105.5 105.6
 105.7 105.8 105.9 106.  106.1 106.2 106.3 106.4 106.5 106.6 106.7 106.8
 106.9 107.  107.1 107.2 107.3 107.4 107.5 107.6 107.7 107.8 107.9 108.
 108.1 108.2 108.3 108.4 108.5 108.6 108.7 108.8 108.9 109.  109.1 109.2
 109.3 109.4 109.5 109.6 109.7 109.8 109.9 110. ]
```

```
In [41]: print(find(data, var=True))
         print(find(data, var=False))
```

```
(57, 465.49729408121016)
(66, 39841.59701492537)
```

可以发现，方差条件下的分割点仍然相较均方误差条件是远离转折点的。

以上几种不同的模拟数据均出现了类似的情况，那我们如果分折线两端对数据进行打乱呢？

5. 打乱的等差数列：

```
In [50]: data = np.linspace(0.1,10, 100)
         np.random.seed(2099)
         np.random.shuffle(data)

         print(data)

         print(find(data, var=True))
         print(find(data, var=False))
```

```
[ 5.9  3.3  4.1  2.8  9.   9.2  1.6  2.   6.7  9.5  6.3  4.4  4.5  7.6
  1.3  1.5  8.4  0.1  8.7  8.9  9.8  0.8  5.7  7.   2.5  7.2  2.6  6.5
  3.9  5.4  3.5  3.   4.8  3.4  0.6  6.   6.6  0.4  8.   1.4  5.5  6.1
  1.8  8.8  1.1  8.2  3.1  1.9  6.9  7.9  4.9  5.8  4.   9.6  2.2  0.2
  8.5  5.3  9.7 10.   0.3  4.6  3.2  0.7  1.2  6.8  4.7  7.1  7.5  2.7
  5.6  8.3  2.9  9.9  1.   2.1  7.8  3.7  5.2  8.6  6.4  3.8  2.4  5.1
  2.3  9.1  7.7  8.1  9.4  4.2  5.   6.2  7.4  0.9  7.3  9.3  0.5  3.6
  1.7  4.3]
(1, 8.409294969901032)
(96, 806.6848958333334)
```

可以看出来此时方差的分割更极端，其实无论是哪个随机种子，影响的只有均方误差情况，方差情况永远是第二个作为分割点。

6. 数据是不对称的，增减性在转折点前后相反的折线，不同增减性的部分分别打乱

```
In [51]: data = np.append(data, np.linspace(10, 1, 10))
         np.random.shuffle(data[100:110])

         print(data)

         print(find(data, var=True))
         print(find(data, var=False))
```

```
[ 5.9  3.3  4.1  2.8  9.   9.2  1.6  2.   6.7  9.5  6.3  4.4  4.5  7.6
  1.3  1.5  8.4  0.1  8.7  8.9  9.8  0.8  5.7  7.   2.5  7.2  2.6  6.5
  3.9  5.4  3.5  3.   4.8  3.4  0.6  6.   6.6  0.4  8.   1.4  5.5  6.1
  1.8  8.8  1.1  8.2  3.1  1.9  6.9  7.9  4.9  5.8  4.   9.6  2.2  0.2
```

```
  8.5  5.3  9.7 10.   0.3  4.6  3.2  0.7  1.2  6.8  4.7  7.1  7.5  2.7
  5.6  8.3  2.9  9.9  1.   2.1  7.8  3.7  5.2  8.6  6.4  3.8  2.4  5.1
  2.3  9.1  7.7  8.1  9.4  4.2  5.   6.2  7.4  0.9  7.3  9.3  0.5  3.6
  1.7  4.3  3.   2.   1.   9.   5.   6.   7.  10.   4.   8. ]
(1, 8.412204359902365)
(103, 890.3446601941747)
```

7. 数据是对称的，增减性在转折点前后相反的折线，不同增减性的部分分别打乱

```
In [52]: data = np.linspace(0.1,10, 100)
         data = np.append(data, np.linspace(10,0.1, 100))

         np.random.shuffle(data[0: 100])
         np.random.shuffle(data[100: 200])

         print(data)

         print(find(data, var=True))
         print(find(data, var=False))
```

```
[ 6.2  4.4  0.9  4.1  0.2  8.2  6.7  9.6  1.6  7.5  3.4  0.6  1.3  5.2
  6.   4.2  3.9  7.4  4.5  3.6  3.7  0.3  8.5  5.9  2.4  0.4  3.5  0.7
  9.   6.8  8.4  3.2  4.8  8.9  3.1  3.3  9.3  2.1  5.3  0.1  8.3  5.
  1.7  9.2  6.3  7.7  2.8  2.5  3.   2.   9.9  6.9  9.8  9.1  7.3  3.8
  2.2  4.3  1.9  6.6  5.5  6.5  1.5  0.5 10.   5.8  1.8  6.4  1.1  8.8
  7.2  2.9  4.6  7.   2.6  8.7  9.5  5.4  5.6  8.6  7.6  8.   8.1  9.7
  1.4  1.2  4.7  5.7  4.9  7.1  4.   2.7  6.1  7.8  5.1  7.9  2.3  9.4
  0.8  1.   5.4  3.   5.6  8.6  3.1  4.9  3.7  5.1  8.8  4.3  9.8  2.9
  7.1  6.4  2.2  9.6  8.5  8.4  3.3  1.1  7.8  2.8  2.4  5.2  8.3  8.7
  4.8  5.5  6.   7.3  7.2  5.3  1.6  1.3  2.6  9.   9.4  8.   9.9  0.2
  0.7  6.3  2.   8.1  9.5  5.7  4.2  6.1  1.9  0.1  4.5  5.   6.9  7.6
  1.5  9.7  3.4  3.8  7.5  8.9  4.4  1.8  3.5  6.2  0.3  4.7  7.4 10.
  8.2  5.9  5.8  1.7  1.2  6.8  2.1  9.2  0.6  2.5  7.9  1.   2.7  6.6
  0.8  0.4  4.6  9.1  3.9  2.3  3.2  7.   4.   7.7  6.7  0.5  3.6  6.5
  0.9  4.1  9.3  1.4]
(1, 8.367692735031945)
(171, 1634.541036499294)
```

8. 数据是单调前后斜率不同的折线，不同斜率分别打乱

```
In [53]: data = np.linspace(1,100, 100)
         data = np.append(data, np.linspace(100.1,110, 100))

         np.random.shuffle(data[0: 100])
         np.random.shuffle(data[100: 200])

         print(data)

         print(find(data, var=True))
         print(find(data, var=False))
```

```
[ 32.   84.   61.   89.   13.   35.   44.   19.   10.    6.   41.   43.
  95.   28.   72.   18.    5.   12.   30.   64.   93.   69.   51.   81.
  96.   66.   71.   58.    3.   38.   23.   15.   37.   52.   76.   60.
   2.   21.   75.   74.   80.   73.   57.   45.   46.   86.   87.   50.
  62.   68.    4.   79.    1.   98.   77.   70.    9.   59.   47.   27.
  48.   97.   31.   20.   85.   40.   33.   39.   56.   65.   82.   90.
  34.   42.   16.   99.   26.   29.  100.   55.    7.   14.   91.   49.
  94.   67.   25.   36.   54.   11.   24.   78.   92.    8.   83.   88.
  63.   17.   53.   22.  107.4 104.4 109.8 105.5 103.3 109.  106.  101.8
 107.  105.8 100.8 102.3 108.2 107.9 101.4 102.4 105.  103.1 108.6 104.6
 108.9 108.3 104.8 104.5 109.2 103.2 104.7 108.  107.2 106.8 106.6 108.4
 109.5 109.4 102.8 105.9 108.8 106.4 102.7 103.6 105.4 101.2 104.9 106.3
 102.1 100.5 104.1 107.5 106.5 100.3 103.9 100.9 105.6 105.3 101.7 101.3
 101.6 102.5 109.1 108.1 107.8 107.3 106.9 105.7 100.6 109.9 102.  100.2
 101.5 101.  102.6 103.8 105.2 103.5 102.9 108.5 105.1 107.6 103.4 108.7
 107.7 107.1 109.3 104.  102.2 103.  106.1 100.7 100.1 109.7 106.2 101.9
 110.  103.7 106.7 104.2 104.3 109.6 100.4 101.1]
(100, 841.5825)
(100, 84158.25)
```

0.4 三、应用

好的现在我们坚定了用均方误差而非方差的选择，这次选用的数据集是经典的波士顿房价数据集，共有 506 个样本，每个样本都有 13 个自变量变量和 1 个因变量：

自变量：

- 城镇人均犯罪率。
- 住宅用地超过 25000 平方英尺的比例。

- 城镇非零售商用土地的比例。
- 是否临近查理斯河。
- 一氧化氮浓度。
- 住宅平均房间数。
- 1940 年之前建成的自用房屋比例。
- 到波士顿五个中心区域的加权距离。
- 接近高速公路的程度。
- 每 10000 美元的不动产税率。
- 城镇师生比例。
- 城镇黑人比例。（相当不政治正确）
- 低收入人口比例

因变量：

- 房屋价格

```
In [105]: from sklearn.datasets import load_boston
          import pandas as pd
          import copy

          boston, target = load_boston(return_X_y=True)
          x = np.array(boston)
          x = np.hstack([x, np.ones((x.shape[0], 1))]) # bias
          y = np.array(target).reshape(len(target), 1)
          print(x.shape)
          print(y.shape)
```

```
(506, 14)
(506, 1)
```

先用正规方程求解线性回归的参数

```
In [106]: np.random.seed(2099)

          index = np.random.permutation(506)
          train_index = index[0: int(0.7*506)]
```

```
        test_index = index[int(0.7*506): 506]

        train_x = x[train_index, :]
        train_y = y[train_index, :]
        test_x = x[test_index, :]
        test_y = y[test_index, :]
```

```
In [107]: beta = np.dot(train_x.T, train_x)
          beta = np.linalg.inv(beta)
          beta = np.dot(beta, train_x.T)
          beta = np.dot(beta, train_y)
          print(beta)
```

```
[[-1.38207153e-01]
 [ 6.02124371e-02]
 [-2.41967408e-03]
 [ 1.38906414e+00]
 [-1.57956187e+01]
 [ 3.26425169e+00]
 [-2.01230038e-03]
 [-1.68034194e+00]
 [ 2.89084987e-01]
 [-9.48385187e-03]
 [-1.03060887e+00]
 [ 8.00698188e-03]
 [-5.77222645e-01]
 [ 4.15481705e+01]]
```

```
In [108]: y_hat = np.dot(test_x, beta)
```

```
In [109]: def mse(y, y_hat):
              tmp = np.power(y-y_hat, 2)
              tmp = np.sum(tmp)
              return tmp

          mse(test_y, y_hat)
```

```
Out[109]: 3906.4556806536502
```

```
In [110]: names = ['CRIM','ZN','INDUS','CHAS','NOX','RM',
                   'AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']
```

```
boston = pd.DataFrame(boston)
boston.columns = names
boston['target'] = target
```

In [111]:
```
def split_value(data, index, variable, discrete, target='target'):
    tmp = data.loc[index]
    n = tmp.shape[0]

    mse = float('inf')
    split = None
    split_value = data[variable][0]

    if discrete:
        values = np.unique(data[variable])
        for value in values:
            index1 = tmp[tmp[variable]==value].index
            index2 = tmp[tmp[variable]!=value].index

            tmp_mse = len(index1)*np.var(tmp.loc[index1][target])
            tmp_mse += len(index2)*np.var(tmp.loc[index2][target])

            if tmp_mse < mse:
                split_value = value
                mse = tmp_mse
    else:
        for value in tmp[variable]:
            index1 = tmp[tmp[variable]<=value].index
            index2 = tmp[tmp[variable]>value].index

            tmp_mse = len(index1)*np.var(tmp.loc[index1][target])
            tmp_mse += len(index2)*np.var(tmp.loc[index2][target])

            if tmp_mse < mse:
                split_value = value
                mse = tmp_mse

    return mse, split_value

split_value(boston, index=train_index, variable='CRIM', discrete=False)
```

Out[111]:
```
(22817.2744320397, 6.65492)
```

```
In [112]: def split_variable(data, index, variable_set, target='target'):
              mse = float('inf')

              to_split_value = None
              to_split_variable = None

              tmp_mse = None
              tmp_value = None
              var_type = None

              for dtype in variable_set:
                  discrete = dtype=='discrete'
                  for variable in variable_set[dtype]:
                      tmp_mse, tmp_value = split_value(data, index, variable,
                                                       discrete, target)
                      #print(str(variable)+': '+str(tmp_value)+': '+str(tmp_mse))

                      if tmp_mse < mse:
                          mse = tmp_mse
                          to_split_value = tmp_value
                          to_split_variable = variable
                          var_type = discrete

              return mse, to_split_variable, to_split_value, var_type

          variable_set = {
              'discrete':['CHAS'],
              'continuous':['CRIM','ZN','INDUS','NOX','RM','AGE',
                            'DIS','RAD','TAX','PTRATIO','B','LSTAT']
          }
          split_variable(boston, index=train_index, variable_set=variable_set)

Out[112]: (15224.583135593219, 'LSTAT', 7.85, False)

In [113]: def build_tree(data, index, variable_set, tree, target, node=1):
              """
              data: all the data
              index: the data on which shall be splited, data won't change in recursion but index
              variable_set: variables to be split
              tree: a dataframe with node,split_variable,split_value,discrete,leaf,left and right
              target: the label variable
```

```
    node: the node of the tree
    """
    tmp = data.iloc[index]
    leaf = len(variable_set[str(node)]['discrete'])==0
    leaf = leaf and (len(variable_set[str(node)]['continuous'])==0)
    leaf = leaf or (len(index)<=10)

    print(str(node)+":"+str(leaf))
    #print(variable_set[str(node)])

    if not leaf:
        mse,variable,value,discrete = split_variable(data, index,
                                                variable_set[str(node)], target)
        #print(variable)

        if variable is not None:
            if discrete:
                variable_set[str(node)]['discrete'].remove(variable)
                variable_set[str(node*2)] = copy.deepcopy(variable_set[str(node)])
                variable_set[str(node*2+1)] = copy.deepcopy(variable_set[str(node)])

                index1 = tmp[tmp[variable] == value].index
                left_prediction = np.mean(data.iloc[index1][target])
                index2 = tmp[tmp[variable] != value].index
                right_prediction = np.mean(data.iloc[index2][target])
            else:
                variable_set[str(node)]['continuous'].remove(variable)
                variable_set[str(node*2)] = copy.deepcopy(variable_set[str(node)])
                variable_set[str(node*2+1)] = copy.deepcopy(variable_set[str(node)])

                index1 = tmp[tmp[variable] <= value].index
                left_prediction = np.mean(data.iloc[index1][target])
                index2 = tmp[tmp[variable] > value].index
                right_prediction = np.mean(data.iloc[index2][target])

            tree.loc[node] = [variable, value, discrete,
                              left_prediction, right_prediction]

            build_tree(data=data, index=index1, variable_set=variable_set,
```

```
                        tree=tree, target='target', node=node*2)
                build_tree(data=data, index=index2, variable_set=variable_set,
                        tree=tree, target='target', node=node*2+1)


tree = {
    'split_variable':[None],
    'split_value':[None],
    'discrete':[None],
    'left_prediction': [None],
    'right_prediction':[None]
}
tree = pd.DataFrame(tree)
variables = {
    'discrete':['CHAS'],
    'continuous':['CRIM','ZN','INDUS','NOX','RM','AGE',
                  'DIS','RAD','TAX','PTRATIO','B','LSTAT']
}
variable_set={
    '1':variables
}
build_tree(boston, train_index, variable_set, tree, target='target')
```

```
1:False
2:False
4:False
8:True
9:False
18:False
36:False
72:True
73:False
146:True
147:False
294:True
295:True
37:False
74:False
148:True
149:False
```

```
298:False
596:False
1192:True
1193:True
597:True
299:True
75:False
150:True
151:False
302:True
303:False
606:True
607:True
19:False
38:True
39:False
78:True
79:False
158:False
316:True
317:False
634:True
635:False
1270:False
2540:True
2541:True
1271:True
159:True
5:False
10:True
11:True
3:False
6:False
12:True
13:False
26:False
52:False
104:True
105:False
```

```
210:False
420:False
840:False
1680:True
1681:False
3362:False
6724:False
13448:True
13449:True
6725:True
3363:True
841:False
1682:True
1683:True
421:True
211:False
422:False
844:False
1688:False
3376:True
3377:True
1689:True
845:True
423:False
846:True
847:False
1694:True
1695:False
3390:True
3391:True
53:True
27:False
54:False
108:True
109:False
218:True
219:False
438:True
439:False
```

```
878:True
879:False
1758:True
1759:False
3518:True
3519:False
55:False
110:False
220:False
440:False
880:False
1760:True
1761:True
881:True
441:True
221:False
442:False
884:True
885:False
1770:True
1771:True
443:True
111:False
222:False
444:True
445:True
223:True
7:False
14:False
28:False
56:True
57:False
114:True
115:True
29:False
58:False
116:True
117:True
59:True
```

```
15:False
30:False
60:True
61:True
31:True
```

In [114]: tree

Out[114]:

	split_variable	split_value	discrete	left_prediction	right_prediction
0	None	NaN	None	NaN	NaN
1	LSTAT	7.85000	False	31.495763	18.367797
2	RM	7.42000	False	28.742857	44.985000
4	DIS	1.35670	False	50.000000	28.071579
9	PTRATIO	18.40000	False	29.756452	24.906061
18	TAX	264.00000	False	32.385000	28.504762
36	AGE	21.10000	False	27.800000	33.531250
73	ZN	0.00000	False	36.600000	32.823077
147	CRIM	0.04932	False	31.325000	33.488889
37	CRIM	0.06724	False	27.121739	30.178947
74	INDUS	1.52000	False	31.700000	25.850000
149	B	396.06000	False	26.975000	23.600000
298	ZN	85.00000	False	26.500000	32.200000
596	NOX	0.46000	False	25.371429	28.475000
75	RAD	4.00000	False	26.533333	31.861538
151	B	353.89000	False	25.650000	32.990909
303	NOX	0.42800	False	23.300000	33.960000
19	CRIM	0.03615	False	29.020000	24.171429
39	B	354.70000	False	27.500000	24.048148
79	TAX	305.00000	False	23.705556	24.733333
158	INDUS	4.86000	False	25.566667	23.333333
317	RAD	3.00000	False	24.500000	23.041667
635	NOX	0.49300	False	23.254545	20.700000
1270	AGE	40.10000	False	23.757143	22.375000
5	PTRATIO	14.70000	False	47.740000	42.230000
3	NOX	0.65500	False	20.219643	13.792647
6	DIS	1.16910	False	50.000000	20.041317
13	PTRATIO	19.60000	False	21.708247	17.731429
26	RM	6.59000	False	21.196629	27.400000
52	INDUS	2.46000	False	26.075000	20.967059
...	...	...	...	...	...

```
844         CHAS    0.00000   True  20.100000  23.000000
1688         RAD    2.00000  False  20.600000  19.683333
423            B  297.09000  False  23.800000  18.652941
847          RAD    3.00000  False  16.150000  18.986667
1695         TAX  304.00000  False  20.257143  17.875000
27             B  352.58000  False  15.600000  18.644898
54          CRIM    0.20746  False   8.100000  15.975000
109           RM    3.86300  False  23.100000  15.600000
219          AGE   48.20000  False  19.900000  15.361111
439        INDUS    8.14000  False  13.533333  15.726667
879          TAX  384.00000  False  18.600000  15.521429
1759         RAD    4.00000  False  15.600000  15.515385
55           AGE   91.90000  False  19.512500  17.011765
110          TAX  334.00000  False  18.481250  20.543750
220           RM    6.09600  False  18.150000  20.800000
440        INDUS    8.14000  False  18.630769  11.900000
880         CRIM    0.62739  False  19.012500  18.020000
221           RM    6.22900  False  20.178571  23.100000
442         CRIM    0.01360  False  18.900000  20.276923
885          RAD    5.00000  False  20.071429  20.516667
111           RM    6.18500  False  15.836364  19.166667
222         CRIM    0.13262  False  19.500000  15.470000
7           CRIM    9.82349  False  16.247619   9.826923
14           AGE   91.40000  False  18.081250  15.119231
28             B  272.21000  False  13.540000  20.145455
57         INDUS   18.10000  False  20.630000  15.300000
29             B  391.71000  False  16.255556  12.562500
58            RM    6.12200  False  17.400000  15.111111
15            RM    6.15200  False   8.568750  11.840000
30           DIS    1.70280  False   7.770000   9.900000

[70 rows x 5 columns]
```

In [115]:
```
def climb(data, tree):
    n = data.shape[0]
    prediction = pd.Series()
    for i in range(n):
        tmp = data.iloc[i]
        leaf = False
        node = 1
```

```
            while not leaf:
                variable = tree.loc[node]['split_variable']
                discrete = tree.loc[node]['discrete']
                if discrete:
                    left = tmp[variable] == tree.loc[node]['split_value']
                else:
                    left = tmp[variable] <= tree.loc[node]['split_value']
                if left:
                    prediction.loc[i] = tree.loc[node]['left_prediction']
                    node = node*2
                else:
                    prediction.loc[i] = tree.loc[node]['right_prediction']
                    node = node*2+1
                leaf = not (node in tree.index)
        return prediction

    prediction = climb(boston.iloc[test_index], tree)
```

```
In [121]: y_hat = np.array(prediction).reshape(len(prediction),1)
          print(y_hat.shape)

(152, 1)
```

```
In [118]: test_y.shape

Out[118]: (152, 1)
```

```
In [123]: mse(test_y, y_hat)

Out[123]: 3743.746454860767
```

回忆线性回归的 mse 是 3906，表明回归树相比线性回归表现要好一些。

```
In [ ]:
```