Lesson 4 CART回归树的建模流程与sklearn参数详解

CART树能同时解决分类问题和回归问题，但由于两类问题的性质存在一定的差异，因此CART树在处理不同类型问题时相应建模流程也略有不同，对应的sklearn中的评估器也是不同的。值得一提的是，虽然单独利用回归树解决问题的场景并不多见，但对于CART回归树的相关方法仍然需要重点掌握，是因为回归树是构建梯度提升树（GBDT，一种集成算法）的唯一基础分类器，并且无论是解决回归类问题还是分类问题。

本节我们将在CART分类树的基础之上详细讨论CART树在处理回归问题时的基本流程，并详细介绍关于CART回归树在sklearn中评估器的的相关参数与使用方法。

In [2]:
```
# 科学计算模块
import numpy as np
import pandas as pd

# 绘图模块
import matplotlib.pyplot as plt

# Scikit-Learn相关模块
# 评估器类
from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor
```

一、CART回归树的基本建模流程

同样，我们先从一个极简的例子来了解CART回归树的基本建模流程，然后再介绍通过数学语言描述得更加严谨的建模流程。

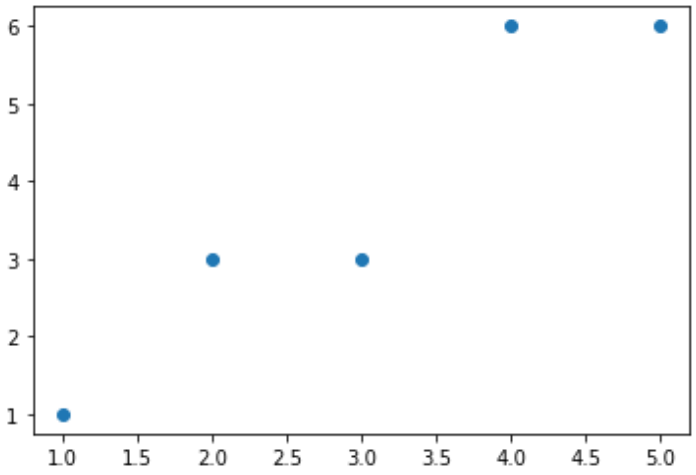
- 数据准备

首先我们创建一个简单的回归数据集如下，该数据集只包含一个特征和一个连续型标签：

数据集A	
x	**y**
1	1
2	3
3	3
4	6
5	6

In [3]:
```
data = np.array([[1, 1], [2, 3], [3, 3], [4, 6], [5, 6]])
plt.scatter(data[:, 0], data[:, 1])
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0xe002848>
```



其中横坐标代表数据集特征，纵坐标代表数据集标签。

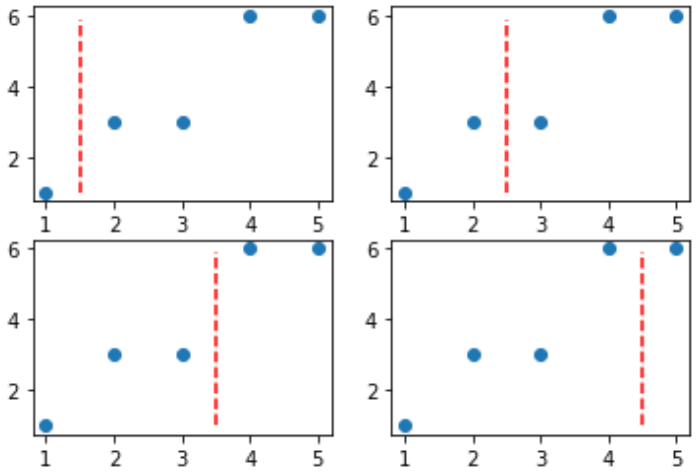
- 节点分裂规则

回归树中寻找切分点的方式和分类树的方式相同，都是逐特征寻找不同取值的中间点作为切分点。对于上述数据集来说，由于只有一个特征，并且总共有5个不同的取值，因此切分点有4个。有几个切分点就有几种数据集划分方式。初始数据集的4个不同的划分方式，可以通过如下方式进行呈现：

In [4]:
```
y_range = np.arange(1, 6, 0.1)
```

In [5]:
```
plt.subplot(221)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 1.5), y_range, 'r--')
plt.subplot(222)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 2.5), y_range, 'r--')
plt.subplot(223)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 3.5), y_range, 'r--')
plt.subplot(224)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 4.5), y_range, 'r--')
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0xe180948>]
```



- 确定分裂准则

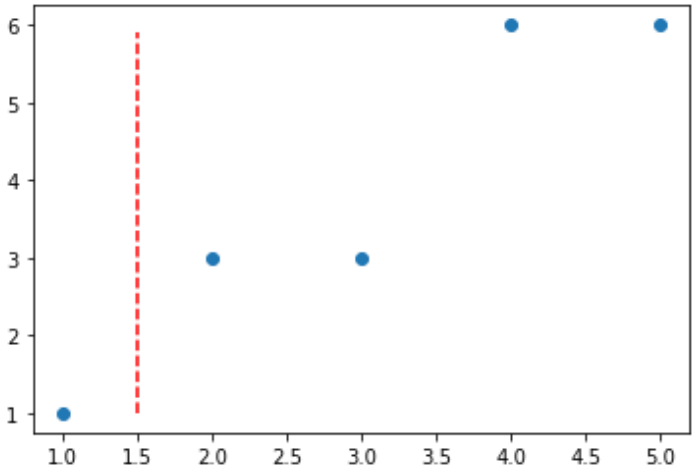
回归树和分类树的分裂准则差异较大，分类树中我们是采用基尼系数或者信息增益来衡量划分后数据集标签不纯度下降情况来挑选最佳划分方式，而在回归树中，则是根据划分之后子数据集MSE下降情况来进行最佳划分方式的挑选。在该过程中，子数据集整体的MSE计算方法也和CART分类树类似，都是先计算每个子集（分支）单独的MSE，然后再通过加权求和的方法来进行计算两个子集整体的MSE。

需要计算MSE，就必须给出一个预测值，然后根据预测值和真实值计算MSE。而CART回归树在进行子数据集的划分之后，会针对每个子数据集给出一个预测值（注意是针对一个子数据集中所有数据给出一个预测值，而不是针对每一个数给出一个预测值），为使子数据集的MSE最小，将子集的标签均值作为整个子集的预测值。

具体计算过程如下，例如对上述第一种划分数据集的情况来说，每个子数据集的预测值和MSE计算结果如下：

In [6]:
```
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 1.5), y_range, 'r--')
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0xe2a9488>]
```



对应划分情况也可以通过如下形式进行表示：

数据集A	
x	**y**
1	1
2	3
3	3
4	6
5	6

TRUE　　划分条件：x <= 1.5　　FALSE

数据集B1	
x	**y**
1	1

数据集B2	
x	**y**
2	3
3	3
4	6
5	6

此时可计算子数据集B1和B2的MSE，首先是两个数据集的预测值，也就是两个数据集的均值：

In [7]:
```
data[0, 1]
```

Out[7]:
```
1
```

In [8]:
```
# B1数据集的预测值
y_1 = np.mean(data[0, 0])
y_1
```

Out[8]:
```
1.0
```

In [9]:
```
data[1: , 1]
```

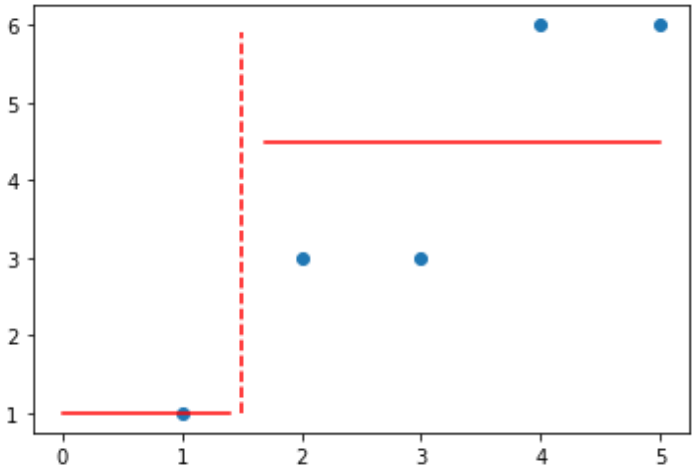
Out[9]:
```
array([3, 3, 6, 6])
```

In [10]:
```
# B2数据集的预测值
y_2 = np.mean(data[1: , 1])
y_2
```

Out[10]:
```
4.5
```

In [11]:
```
# 模型预测结果
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 1.5), y_range, 'r--')
plt.plot(np.arange(0, 1.5, 0.1), np.full_like(np.arange(0, 1.5, 0.1), y_1), 'r-')
plt.plot(np.arange(1.7, 5.1, 0.1), np.full_like(np.arange(1.7, 5.1, 0.1), y_2), 'r-')
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0xe397f48>]
```



然后计算两个子集的MSE：

In [12]:
```
# B1的MSE
mse_b1 = 0
```

In [13]:
```
# B2的MSE
mse_b2 = np.power(data[1: , 1] - 4.5, 2).sum() / 4
mse_b2
```

Out[13]:
```
2.25
```

然后和CART分类树一样，以各子集所占全集的样本比例为权重，通过加权求和的方式计算两个子集整体的MSE：

In [14]:
```
mse_b = 1/5 * mse_b1 + 4/5 * mse_b2
mse_b
```

Out[14]:
```
1.8
```

而父节点的MSE为：

In [15]:
```
data[: , 1].mean()
```

Out[15]:
```
3.8
```

In [16]:
```
mse_a = np.power(data[: , 1] - data[: , 1].mean(), 2).sum() / data[: , 1].size
mse_a
```

Out[16]:
```
3.7599999999999993
```

因此，本次划分所降低的MSE为：

In [17]:
```
mse_a - mse_b
```

Out[17]:
```
1.9599999999999993
```

即为该种划分方式的最终评分。当然，我们要以相似的流程计算其他几种划分方式的评分，然后从中挑选能够最大程度降低MSE的划分方式，基本流程如下：

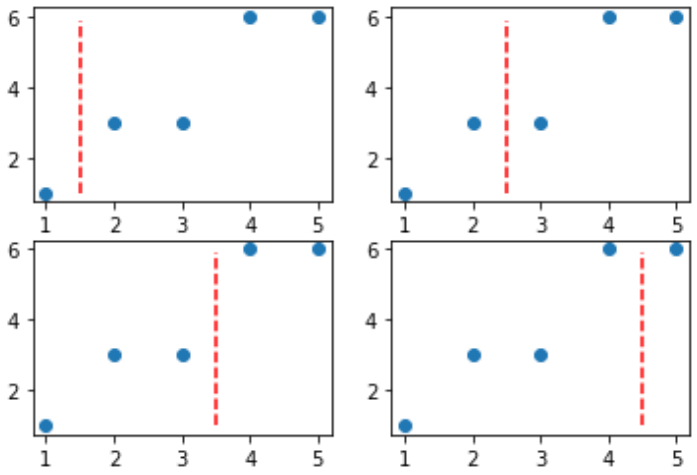
In [18]:
```
impurity_decrease = []

for i in range(4):
    # 寻找切分点
    splitting_point = data[i: i+2 , 0].mean()

    # 进行数据集切分
    data_b1 = data[data[:, 0] <= splitting_point]
    data_b2 = data[data[:, 0] > splitting_point]

    # 分别计算两个子数据集的MSE
    mse_b1 = np.power(data_b1[: , 1] - data_b1[: , 1].mean(), 2).sum() / data_b1[: ,
    mse_b2 = np.power(data_b2[: , 1] - data_b2[: , 1].mean(), 2).sum() / data_b2[: ,

    # 计算两个子数据集整体的MSE
    mse_b = data_b1[: , 1].size/data[: , 1].size * mse_b1 + data_b2[: , 1].size/data[
    #mse_b = mse_b1 + mse_b2

    # 计算当前划分情况下MSE下降结果
    impurity_decrease.append(mse_a - mse_b)
```

In [19]:
```
impurity_decrease
```

Out[19]:
```
[1.9599999999999993, 2.1599999999999993, 3.226666666666666, 1.209999999999999]
```

In [20]:
```
plt.subplot(221)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 1.5), y_range, 'r--')
plt.subplot(222)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 2.5), y_range, 'r--')
plt.subplot(223)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 3.5), y_range, 'r--')
plt.subplot(224)
plt.scatter(data[:, 0], data[:, 1])
plt.plot(np.full_like(y_range, 4.5), y_range, 'r--')
```

Out[20]: [<matplotlib.lines.Line2D at 0xe453508>]



根据最终结果能够看出，第三种划分情况能够最大程度降低MSE，即此时树模型的第一次生长情况如下：

数据集A

x	y
1	1
2	3
3	3
4	6
5	6

TRUE　　划分条件：x <= 3.5　　FALSE

数据集B1

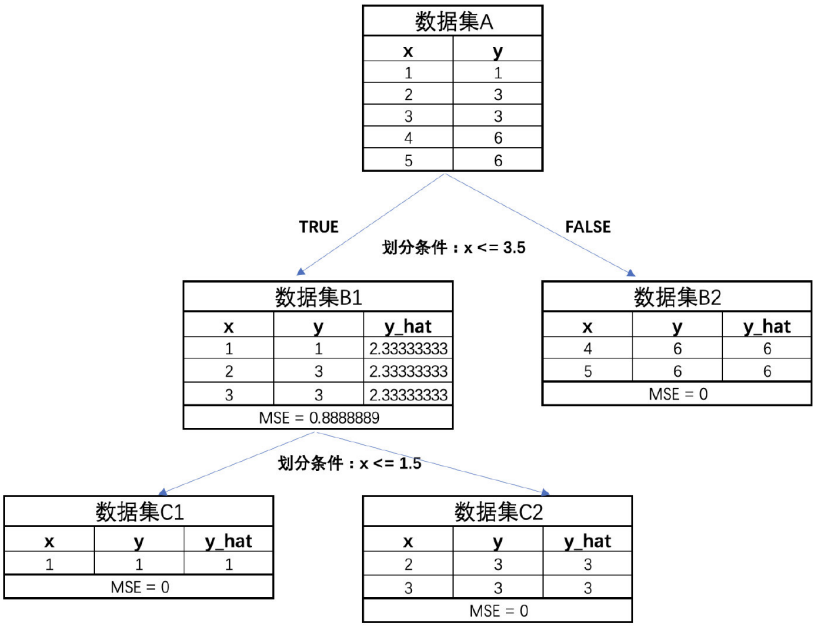
x	y	y_hat
1	1	2.3333333
2	3	2.3333333
3	3	2.3333333
MSE = 0.8888889		

数据集B2

x	y	y_hat
4	6	6
5	6	6
MSE = 0		

- 进行多轮迭代

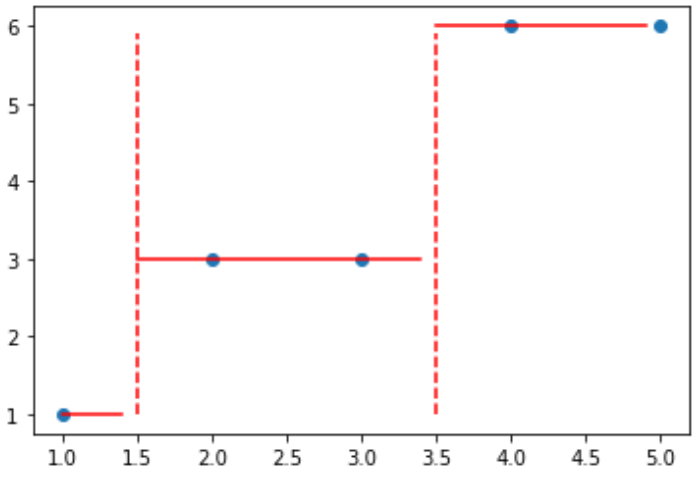
当然，和CART分类树一样，接下来，我们就能够进一步围绕B1和B2进行进一步划分。此时B2的MSE已经为0，因此无需再进行划分，而B1的MSE为0.88，还可以进一步进行划分。当然B1的划分过程也和A数据集的划分过程一致，寻找能够令子集MSE下降最多的方式进行切分。不过由于B1数据集本身较为简单，通过观察不难发现，我们可以以x<= 1.5作为划分条件对其进行切分，进一步切分出来的子集的MSE都将取值为0。



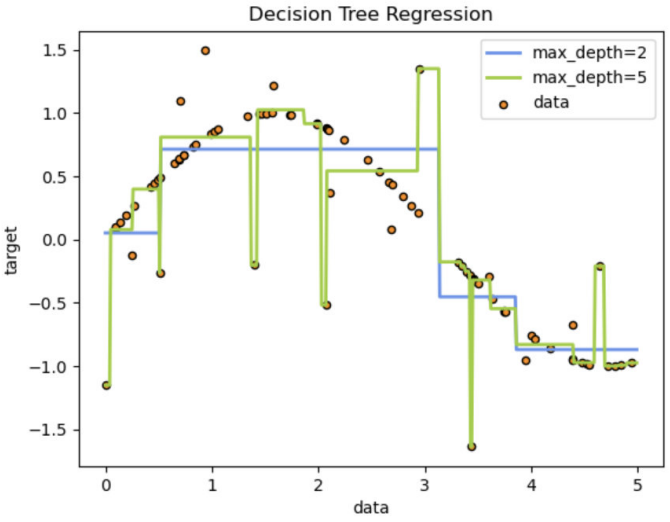
此外，我们也可以观察当回归树生长了两层之后，相关规则对原始数据集的划分情况：

In [21]:
```
# 数据分布
plt.scatter(data[:, 0], data[:, 1])

# 两次划分
plt.plot(np.full_like(y_range, 3.5), y_range, 'r--')
plt.plot(np.full_like(y_range, 1.5), y_range, 'r--')

# 预测结果
plt.plot(np.arange(1, 1.5, 0.1), np.full_like(np.arange(1, 1.5, 0.1), 1), 'r-')
plt.plot(np.arange(1.5, 3.5, 0.1), np.full_like(np.arange(1.5, 3.5, 0.1), 3), 'r-')
plt.plot(np.arange(3.5, 5, 0.1), np.full_like(np.arange(3.5, 5, 0.1), 6), 'r-')
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0xe5da688>]
```



不难发现，回归树的对标签的预测，实际上是一种"分区定值"的预测，建模过程的实际表现是对样本进行划分、然后每个区间给定一个预测值，并且树的深度越深、对样本空间的划分次数就越多、样本空间就会被分割成更多的子空间。在sklearn的说明文档中也有相关例子：



- 回归树的预测过程

而一旦当模型已经构建完成后，回归树的预测过程其实和分类树非常类似，新数据只要根据划分规则分配到所属样本空间，则该空间模型的预测结果就是该数据的预测结果。

至此，我们就在一个极简的数据集上完成了CART回归树的构建。不难发现，回归树和分类树的构建过程大致相同、迭代过程也基本一致，我们可以将其视作同一种建模思想的两种不同实现形式。

二、CART回归树的Scikit-Learn实现方法

1.CART回归树的sklearn快速实现

接下来，我们尝试在sklearn中调用回归树评估器围绕上述数据集进行建模，并对上述过程进行简单验证。回归树也是在tree模块下，我们可以通过如下方式进行导入：

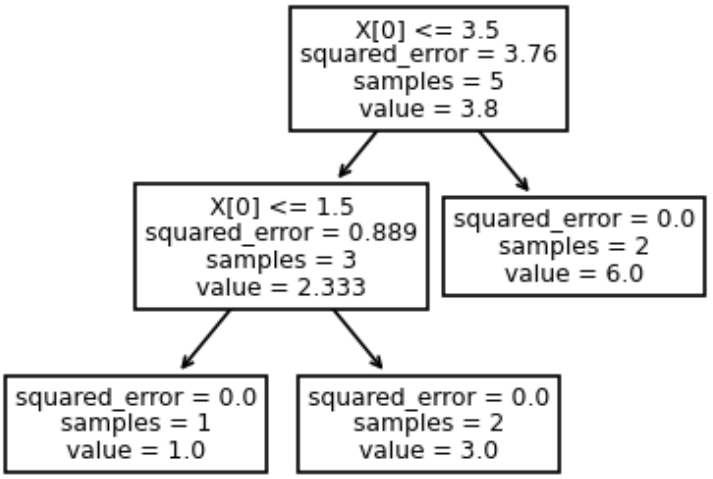
In [26]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
```

然后进行模型训练：

In [27]:
```
clf = DecisionTreeRegressor().fit(data[:, 0].reshape(-1, 1), data[:, 1])
```

In [35]:
```
# 同样可以借助tree.plot_tree进行结果的可视化呈现
plt.figure(figsize=(4, 3), dpi=120)
tree.plot_tree(clf)
```

Out[35]:
```
[Text(0.6, 0.8333333333333334, 'X[0] <= 3.5\nsquared_error = 3.76\nsamples = 5\nvalue
= 3.8'),
 Text(0.4, 0.5, 'X[0] <= 1.5\nsquared_error = 0.889\nsamples = 3\nvalue = 2.333'),
 Text(0.2, 0.16666666666666666, 'squared_error = 0.0\nsamples = 1\nvalue = 1.0'),
 Text(0.6, 0.16666666666666666, 'squared_error = 0.0\nsamples = 2\nvalue = 3.0'),
 Text(0.8, 0.5, 'squared_error = 0.0\nsamples = 2\nvalue = 6.0')]
```



发现和我们手动实现过程一致。

2.CART回归树评估器的参数解释

尽管CART回归树和分类树是由不同评估器实现相关过程，但由于两种模型基本理论一致，因此两种不同评估器的参数也大都一致。

其中大多数参数与CART分类树相同，此处重点讲解criterion参数取值。criterion是节点分裂准则，对于CART分类树来说默认基尼系数、可选信息增益，而对于CART回归树来说可以选择"squared_error"（平方误差），"absolute_error"（绝对误差），“friedman_mse”（梯度提升树的指标）以及"poisson"（泊松偏差）。接下来我们就这几个参数不同取值进行介绍：

- criterion='squared_error'情况

当criterion取值为squared_error时是计算mse，误差平方和再除以样本总数，其基本计算流程与上述手动实现过程层类似。

- criterion='absolute_error'情况

和MSE不同，absolute_error(也就是MAE)实际上计算的是预测值和真实值的差值的绝对值再除以样本总数，即可以通过如下公式计算得出：

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|(y_i - \hat{y}_i)|$$

也就是说，MSE是基于预测值和真实值之间的欧式距离进行的计算，而MAE则是基于二者的街道距离进行的计算，很多时候，MSE也被称为L2损失，而MAE则被称为L1损失。

需要注意的是，当criterion取值为MAE时，为了让每一次划分时子集内的MAE值最小，此时每个子集的模型预测值就不再是均值，而是中位数。

> 再次强调，对于回归树来说，criterion的取值其实决定了两个方面，其一是决定了损失值的计算方式、其二是决定了每一个数据集的预测值的计算方式——数据

> 集的预测值要求criterion取值最小，如果criterion=mse，则数据集的预测值要求在当前数据情况下mse取值最小，此时应该以数据集的标签的均值作为预测值；而如果criterion=mse，则数据集的预测值要求在当前数据情况下mae取值最小，此时应该以数据集标签的中位数作为预测值。

并且一般来说，如果希望模型对极端值（非常大或者非常小的值，也被称为离群值）的忍耐程度比较高，整体建模过程不受极端值影响，可以考虑使用mae参数（就类似于中位数会更少的受到极端值的影响），此时模型一般不会为极端值单独设置规则。而如果希望模型具备较好的识别极端值的能力，则可以考虑使用mse参数，此时模型会更大程度受到极端值影响（就类似于均值更容易受到极端值影响），更大概率会围绕极端值单独设置规则，从而帮助建模者对极端值进行识别。

- criterion='friedman_mse'情况

friedman_mse是一种基于mse的改进型指标，是由GBDT（梯度提升树，一种集成算法）的提出者friedman所设计的一种残差计算方法，是sklearn中梯度提升树默认的criterion取值，对于单独的树决策树模型一般不推荐使用。

- criterion='poisson'情况

$$泊松偏差 = 2\sum(y_i log(\frac{y_i}{\hat{y_i}}) - (y_i - \hat{y_i}))$$

泊松偏差则是适用于一个特殊场景的：当需要预测的标签全部为正整数时，标签的分布可以被认为是类似于泊松分布的。正整数预测在实际应用中非常常见，比如预测点击量、预测客户/离职人数、预测销售量等。

另外，当我们选择不同的criterion之后，决策树的feature_importances_也会随之变化，因为在sklearn当中，feature_importances_是特征对criterion下降量的总贡献量，因此不同的criterion可能得到不同的特征重要性。

In [ ]: