

《机器学习实战》学习笔记（九）：树回归

原创 我是管小亮 2019-09-27 09:51:05 1773 收藏 5

版权

分类专栏: Machine Learning 文章标签: 机器学习 机器学习实战 树回归 CART 树剪枝

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- <https://blog.csdn.net/TeFuirnever/article/details/99701256>

GitHub代码地址:

- <https://github.com/TeFuirnever/Machine-Learning-in-Action>

目录

欢迎关注WX公众号：【程序员管小亮】

本章内容

- 1、复杂数据的回归问题
- 2、ID3 算法与 CART 算法
- 3、连续和离散型特征的树的构建
- 4、将CART 算法用于回归
- 5、树剪枝
 - 1) 预剪枝
 - 2) 后剪枝
- 6、模型树
- 7、示例：树回归与标准回归的比较
- 8、使用 Python 的 Tkinter 库创建 GUI
- 9、总结

参考文章

本章内容

- CART算法
- 回归与模型树
- 树剪枝算法
- Python中GUI的使用

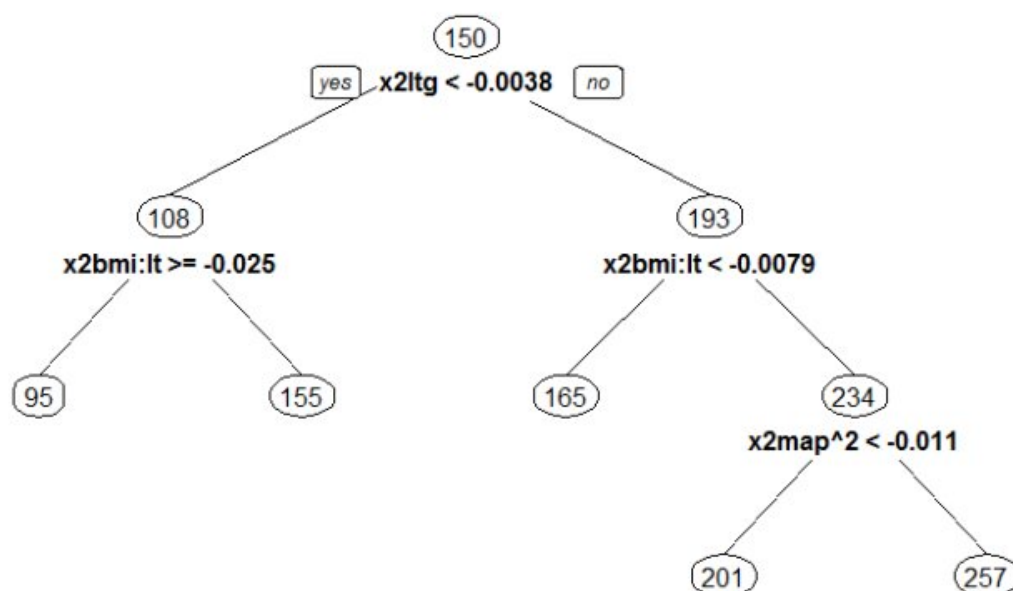
1、复杂数据的回归问题

在《机器学习实战》学习笔记（八）：预测数值型数据 - 回归 中，包含了一些强大的方法，如：岭回归、lasso、前向逐步回归等，但这些方法创建的模型需要拟合所有的样本点（局部加权线性回归除外）。当数

据拥有 **众多特征并且特征之间关系十分复杂** 时，构建全局模型的想法就显得太难了，也略显笨拙。而且，实际生活中很多问题都是 **非线性的**，不可能使用全局线性模型来拟合任何数据。

一种可行的方法是将数据集切分成很多份易建模的数据，然后利用线性回归技术来建模。如果首次切分后仍然难以拟合线性模型就继续切分。在这种切分方式下**，树结构** 和 **回归法** 就相当有用。

CART (Classification And Regression Trees, 分类回归树) 就是一种树构建算法。该算法既可以用于 **分类** 还可以用于 **回归**，因此非常值得学习。



<https://blog.csdn.net/TeFuimever>

2、ID3 算法与 CART 算法

《机器学习实战》学习笔记（三）：决策树 中使用决策树来进行分类。决策树不断将数据切分成小数据集，直到所有目标变量完全相同，或者数据不能再切分为止。决策树是一种 **贪心算法**，它要在给定时间内做出最佳选择，但并不关心能否达到全局最优。

树回归

优点：可以对复杂和非线性的数据建模。

缺点：结果不易理解。

适用数据类型：数值型和标称型数据。

《机器学习实战》学习笔记（三）：决策树 中使用的树构建算法是ID3。ID3的做法是 **每次选取当前最佳的特征来分割数据，并按照该特征的所有可能取值来切分**。也就是说，如果一个特征有4种取值，那么数据将被切成4份。一旦按某特征切分后，该特征在之后的算法执行过程中将不会再起作用，所以有观点认为这种 **切分方式过于迅速**。另外一种方法是二元切分法，即每次把数据集切成两份。如果数据的某特征值等于切分所要求的值，那么这些数据就进入树的左子树，反之则进入树的右子树。

除了切分过于迅速外，ID3算法还存在另一个问题，它 **不能直接处理连续型特征**。只有事先将连续型特征转换成离散型，才能在ID3算法中使用，但这种转换过程会破坏连续型变量的内在性质。而使用二元切分法则易于对树构建过程进行调整以处理连续型特征，具体的处理方法是：如果特征值大于给定值就走左子树，否则就走右子树。另外，二元切分法也节省了树的构建时间，但这点意义也不是特别大，因为这些树构建一般是离线完成，时间并非需要重点关注的因素。

CART是十分著名且广泛记载的树构建算法，它使用二元切分来处理连续型变量。对CART稍作修改就可以处理回归问题。回归树与分类树的思路类似，但叶节点的数据类型不是离散型，而是连续型。

树回归的一般方法

- (1) 收集数据：采用任意方法收集数据。
- (2) 准备数据：需要数值型的数据，标称型数据应该映射成二值型数据。
- (3) 分析数据：绘出数据的二维可视化显示结果，以字典方式生成树。
- (4) 训练算法：大部分时间都花费在叶节点树模型的构建上。
- (5) 测试算法：使用测试数据上的R²值来分析模型的效果。
- (6) 使用算法：使用训练出的树做预测，预测结果还可以用来做很多事情

3、连续和离散型特征的树的构建

在树的构建过程中，需要解决多种类型数据的存储问题。这里使用一部字典来存储树的数据结构，该字典将包含以下4个元素。

- 待切分的特征。
- 待切分的特征值。
- 右子树。当不再需要切分的时候，也可以是单个值。
- 左子树。与右子树类似。

CART算法只做二元切分，所以可以固定树的数据结构。树包含左键和右键，可以存储另一棵子树或者单个值。字典还包含特征和特征值这两个键，它们给出切分算法所有的特征和特征值。

先来看一个简单的例子，创建一个简单的矩阵，然后按照指定列的某个值切分该矩阵。

```
1  import numpy as np
2
3  # 函数说明: 根据给定特征和特征值, 通过数组过滤的方式切分数据集
4  """
5  Parameters:
6      dataSet - 数据集
7      feature - 待切分的特征
8      value - 特征的某个值
9  """
10 def binSplitDataSet(dataSet, feature, value):
11     mat0 = dataSet[np.nonzero(dataSet[:,feature] > value)[0],:]
12     mat1 = dataSet[np.nonzero(dataSet[:,feature] <= value)[0],:]
13     return mat0, mat1
14
15 if __name__ == '__main__':
16     testMat = np.mat(np.eye(4))
17     print("testMat: ", testMat)
18     mat0, mat1 = binSplitDataSet(testMat, 1, 0.5)
19     print("mat0: ", mat0)
20     print("mat1: ", mat1)
21
```

```
testMat: [[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
mat0: [[ 0.  1.  0.  0.]]
mat1: [[ 1.  0.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

4、将CART 算法用于回归

要对数据的复杂关系建模，这里借用树结构来帮助切分数据，那么如何实现数据的切分呢？怎么才能知道是否已经充分切分呢？这些问题的答案取决于叶节点的建模方式。回归树假设叶节点是常数值，这种策略认为数据中的复杂关系可以用树结构来概括。

为成功构建以分段常数为叶节点的树，需要度量出数据的一致性。使用树进行分类，会在给定节点时计算数据的混乱度。那么如何计算连续型数值的混乱度呢？事实上，在数据集上计算混乱度是非常简单的。首先计算所有数据的均值，然后计算每条数据的值到均值的差值。为了对正负差值同等看待，一般使用绝对值或平方值来代替上述差值。

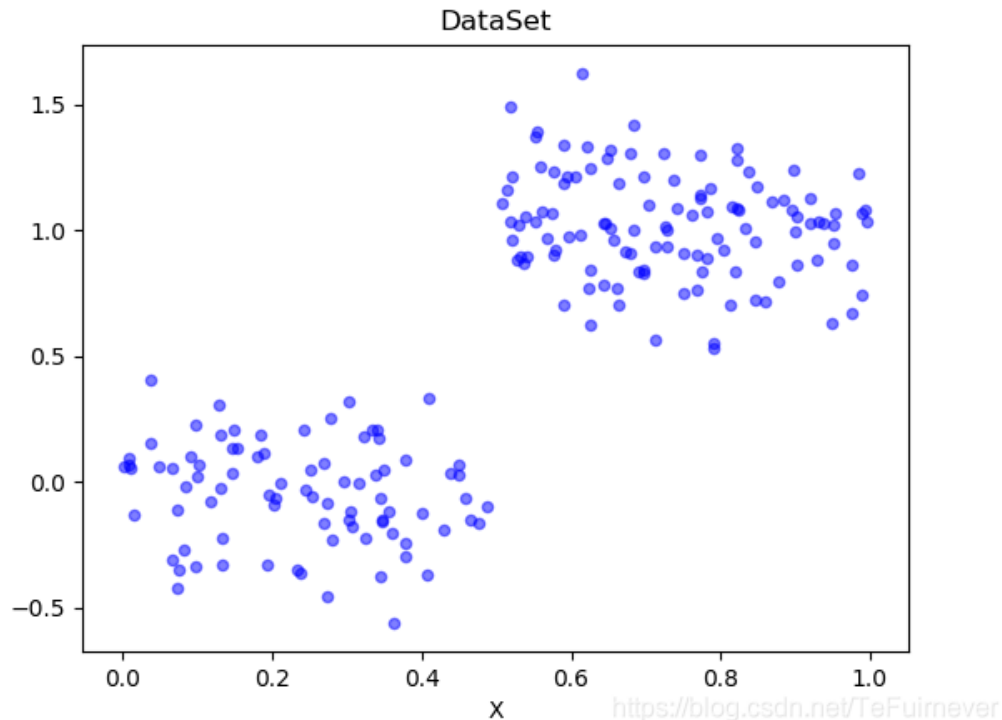
```
1  from numpy import *
2  import matplotlib.pyplot as plt
3
4  # 函数说明: 加载数据
5  """
6  Parameters:
7      filename - 文件名
8  """
9  def loadDataSet(fileName):
10     dataMat = []
11     fr = open(fileName)
12     for line in fr.readlines():
13         curLine = line.strip().split('\t')
14         fltLine = list(map(float, curLine))
15         dataMat.append(fltLine)
16     return dataMat
17
18 # 函数说明: 绘制数据集分布
19 """
20 Parameters:
21     filename - 文件名
22 """
23 def plotDataSet(filename):
24     dataMat = loadDataSet(filename)
25     n = len(dataMat) # 数据个数
26     xcord = []
27     ycord = [] # 样本点
28     for i in range(n):
29         xcord.append(dataMat[i][0])
30         ycord.append(dataMat[i][1]) # 样本点
31     fig = plt.figure()
32     ax = fig.add_subplot(111) # 添加subplot
33     ax.scatter(xcord, ycord, s=20, c='blue', alpha=.5) # 绘制样本点
34     plt.title('DataSet') # 绘制title
35     plt.xlabel('X')
36     plt.show()
```

```

35
36
37 if __name__ == '__main__':
38     filename = 'ex00.txt'
39     plotDataSet(filename)
40

```

基于CART算法构建回归树的简单数据集



下面利用这个数据集来测试一下CART算法。

```

1  import numpy as np
2
3  # 函数说明: 加载数据
4  """
5  Parameters:
6      fileName - 文件名
7  """
8  def loadDataSet(fileName):
9      dataMat = []
10     fr = open(fileName)
11     for line in fr.readlines():
12         curLine = line.strip().split('\t')
13         fltLine = list(map(float, curLine)) # 转化为float类型
14         dataMat.append(fltLine)
15     return dataMat
16
17 # 函数说明: 根据特征切分数据集
18 """
19 Parameters:
20     dataSet - 数据集
21     feature - 带切分的特征
22     value - 该特征的值
23 """
24 def binSplitDataSet(dataSet, feature, value):
25     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0], :]
26     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0], :]
27     return mat0, mat1
28

```

```

25     mat1 = dataSet[np.nonzero(dataSet[:,feature] <= value)[0],:]
26     return mat0, mat1
27
28 # 函数说明: 生成叶结点
29 """
30 Parameters:
31     dataSet - 数据集合
32 """
33 def regLeaf(dataSet):
34     return np.mean(dataSet[:, -1])
35
36 # 函数说明: 误差估计函数
37 """
38 Parameters:
39     dataSet - 数据集合
40 """
41 def regErr(dataSet):
42     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
43
44 # 函数说明: 找到数据的最佳二元切分方式函数
45 """
46 Parameters:
47     dataSet - 数据集合
48     leafType - 生成叶结点
49     regErr - 误差估计函数
50     ops - 用户定义的参数构成的元组
51 """
52 def chooseBestSplit(dataSet, leafType = regLeaf, errType = regErr, ops = (1,4)):
53     import types
54     #tolS允许的误差下降值, tolN切分的最少样本数
55     tolS = ops[0]; tolN = ops[1]
56     #如果当前所有值相等, 则退出。(根据set的特性)
57     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
58         return None, leafType(dataSet)
59     #统计数据集合的行m和列n
60     m, n = np.shape(dataSet)
61     #默认最后一个特征为最佳切分特征, 计算其误差估计
62     S = errType(dataSet)
63     #分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
64     bestS = float('inf'); bestIndex = 0; bestValue = 0
65     #遍历所有特征列
66     for featIndex in range(n - 1):
67         #遍历所有特征值
68         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
69             #根据特征和特征值切分数据集
70             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
71             #如果数据少于tolN, 则退出
72             if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN): continue
73             #计算误差估计
74             newS = errType(mat0) + errType(mat1)
75             #如果误差估计更小, 则更新特征索引值和特征值
76             if newS < bestS:
77                 bestIndex = featIndex
78                 bestValue = splitVal
79                 bestS = newS
80             #如果误差减少不大则退出
81             if (S - bestS) < tolS:
82                 return None, leafType(dataSet)
83     #根据最佳的切分特征和特征值切分数据集

```

```

82     mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
83     #如果切分出的数据集很小则退出
84     if (np.shape(mat0)[0] < to1N) or (np.shape(mat1)[0] < to1N):
85         return None, leafType(dataSet)
86     #返回最佳切分特征和特征值
87     return bestIndex, bestValue
88
89 if __name__ == '__main__':
90     myDat = loadDataSet('ex00.txt')
91     myMat = np.mat(myDat)
92     feat, val = chooseBestSplit(myMat, regLeaf, regErr, (1, 4))
93     print(feat)
94     print(val)

```

```

0
0.48813

```

可以看到，最佳切分特征为第1列特征，最佳切分特征值为0.48813，这个特征值怎么选出来的？

就是根据误差估计的大小，而选择的这个特征值可以使误差最小化。接下来利用选出的这两个变量创建回归树了，首先根据切分的特征和特征值切分出两个数据集，然后将两个数据集分别用于左子树的构建和右子树的构建，直到无法找到切分的特征为止。代码如下：

```

1  import numpy as np
2
3  # 函数说明: 加载数据
4  """
5  Parameters:
6      fileName - 文件名
7  """
8  def loadDataSet(fileName):
9
10     dataMat = []
11     fr = open(fileName)
12     for line in fr.readlines():
13         curLine = line.strip().split('\t')
14         fltLine = list(map(float, curLine))           #转化为float类型
15         dataMat.append(fltLine)
16     return dataMat
17
18 # 函数说明: 根据特征切分数据集
19 """
20 Parameters:
21     dataSet - 数据集
22     feature - 带切分的特征
23     value - 该特征的值
24 """
25 def binSplitDataSet(dataSet, feature, value):
26     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0], :]
27     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0], :]
28     return mat0, mat1
29
30 # 函数说明: 生成叶结点
31 """
32 Parameters:
33     dataSet - 数据集
34 """

```

```

33 .....
34 def regLeaf(dataSet):
35     return np.mean(dataSet[:, -1])
36
37 # 函数说明: 误差估计函数
38 """
39 Parameters:
40     dataSet - 数据集合
41 """
42 def regErr(dataSet):
43     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
44
45 # 函数说明: 找到数据的最佳二元切分方式函数
46 """
47 Parameters:
48     dataSet - 数据集合
49     leafType - 生成叶结点
50     regErr - 误差估计函数
51     ops - 用户定义参数构成的元组
52 """
53 def chooseBestSplit(dataSet, leafType = regLeaf, errType = regErr, ops = (1, 4)):
54     import types
55     # toLS 允许的误差下降值, toLN 切分的最少样本数
56     toLS = ops[0]; toLN = ops[1]
57     # 如果当前所有值相等, 则退出。(根据set的特性)
58     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
59         return None, leafType(dataSet)
60     # 统计数据集合的行m和列n
61     m, n = np.shape(dataSet)
62     # 默认最后一个特征为最佳切分特征, 计算其误差估计
63     S = errType(dataSet)
64     # 分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
65     bestS = float('inf'); bestIndex = 0; bestValue = 0
66     # 遍历所有特征列
67     for featIndex in range(n - 1):
68         # 遍历所有特征值
69         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
70             # 根据特征和特征值切分数据集
71             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
72             # 如果数据少于toLN, 则退出
73             if (np.shape(mat0)[0] < toLN) or (np.shape(mat1)[0] < toLN): continue
74             # 计算误差估计
75             newS = errType(mat0) + errType(mat1)
76             # 如果误差估计更小, 则更新特征索引值和特征值
77             if newS < bestS:
78                 bestIndex = featIndex
79                 bestValue = splitVal
80                 bestS = newS
81     # 如果误差减少不大则退出
82     if (S - bestS) < toLS:
83         return None, leafType(dataSet)
84     # 根据最佳的切分特征和特征值切分数据集
85     mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
86     # 如果切分出的数据集很小则退出
87     if (np.shape(mat0)[0] < toLN) or (np.shape(mat1)[0] < toLN):
88         return None, leafType(dataSet)
89     # 返回最佳切分特征和特征值
90     return bestIndex, bestValue

```



```

90 # 函数说明: 树构建函数
91 """
92 Parameters:
93     dataSet - 数据集
94     leafType - 建立叶结点的函数
95     errType - 误差计算函数
96     ops - 包含树构建所有其他参数的元组
97 """
98 def createTree(dataSet, leafType = regLeaf, errType = regErr, ops = (1, 4)):
99     # 选择最佳切分特征和特征值
100     feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
101     # 如果没有特征, 则返回特征值
102     if feat == None: return val
103     # 回归树
104     retTree = {}
105     retTree['spInd'] = feat
106     retTree['spVal'] = val
107     # 分成左数据集和右数据集
108     lSet, rSet = binSplitDataSet(dataSet, feat, val)
109     # 创建左子树和右子树
110     retTree['left'] = createTree(lSet, leafType, errType, ops)
111     retTree['right'] = createTree(rSet, leafType, errType, ops)
112     return retTree
113
114 if __name__ == '__main__':
115     myDat = loadDataSet('ex00.txt')
116     myMat = np.mat(myDat)
117     print(createTree(myMat))

```

```
{'spInd': 0, 'spVal': 0.48813, 'left': 1.0180967672413792, 'right': -0.044650285714285719}
```

从上图可知，这棵树只有两个叶结点。

下面换一个复杂一点的数据集，分段常数数据集。

```

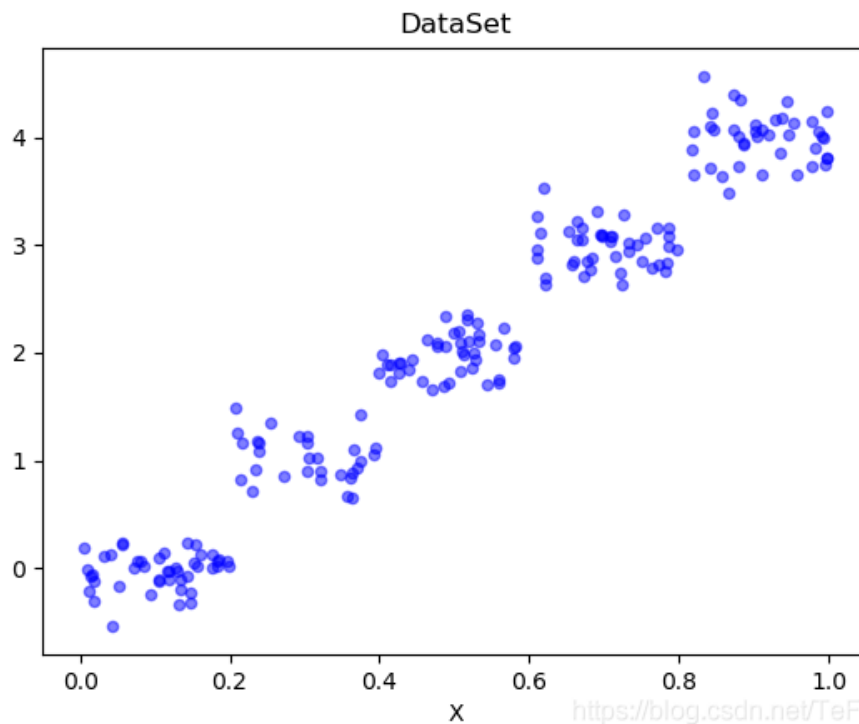
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # 函数说明: 加载数据
5 """
6 Parameters:
7     filename - 文件名
8 """
9 def loadDataSet(fileName):
10     dataMat = []
11     fr = open(fileName)
12     for line in fr.readlines():
13         curLine = line.strip().split('\t')
14         fltLine = list(map(float, curLine)) # 转化为float类型
15         dataMat.append(fltLine)
16     return dataMat
17
18 # 函数说明: 绘制数据集
19 """
20 Parameters:
21     filename - 文件名

```

```

21 """
22 def plotDataSet(filename):
23     dataMat = loadDataSet(filename)           #加载数据集
24     n = len(dataMat)                           #数据个数
25     xcord = []; ycord = []                     #样本点
26     for i in range(n):
27         xcord.append(dataMat[i][1]); ycord.append(dataMat[i][2]) #样本点
28     fig = plt.figure()
29     ax = fig.add_subplot(111)                  #添加subplot
30     ax.scatter(xcord, ycord, s = 20, c = 'blue', alpha = .5) #绘制样本点
31     plt.title('DataSet')                       #绘制title
32     plt.xlabel('X')
33     plt.show()
34
35 if __name__ == '__main__':
36     filename = 'ex0.txt'
37     plotDataSet(filename)
38

```



```

1 import numpy as np
2
3 # 函数说明: 加载数据
4 """
5 Parameters:
6     fileName - 文件名
7 """
8 def loadDataSet(fileName):
9
10     dataMat = []
11     fr = open(fileName)
12     for line in fr.readlines():
13         curLine = line.strip().split('\t')
14         fltLine = list(map(float, curLine))           #转化为float类型
15         dataMat.append(fltLine)

```

```

16         return dataMat
17
18 # 函数说明: 根据特征切分数据集
19 """
20 Parameters:
21     dataSet - 数据集
22     feature - 带切分的特征
23     value - 该特征的值
24 """
25 def binSplitDataSet(dataSet, feature, value):
26     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0], :]
27     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0], :]
28     return mat0, mat1
29
30 # 函数说明: 生成叶结点
31 """
32 Parameters:
33     dataSet - 数据集
34 """
35 def regLeaf(dataSet):
36     return np.mean(dataSet[:, -1])
37
38 # 函数说明: 误差估计函数
39 """
40 Parameters:
41     dataSet - 数据集
42 """
43 def regErr(dataSet):
44     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
45
46 # 函数说明: 找到数据的最佳二元切分方式函数
47 """
48 Parameters:
49     dataSet - 数据集
50     leafType - 生成叶结点
51     regErr - 误差估计函数
52     ops - 用户定义的参数构成的元组
53 """
54 def chooseBestSplit(dataSet, leafType = regLeaf, errType = regErr, ops = (1, 4)):
55     import types
56     #tolS允许的误差下降值, tolN切分的最少样本数
57     tolS = ops[0]; tolN = ops[1]
58     #如果当前所有值相等, 则退出。(根据set的特性)
59     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
60         return None, leafType(dataSet)
61     #统计数据集的行m和列n
62     m, n = np.shape(dataSet)
63     #默认最后一个特征为最佳切分特征, 计算其误差估计
64     S = errType(dataSet)
65     #分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
66     bestS = float('inf'); bestIndex = 0; bestValue = 0
67     #遍历所有特征列
68     for featIndex in range(n - 1):
69         #遍历所有特征值
70         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
71             #根据特征和特征值切分数据集
72             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
73             #如果数据少于tolN, 则退出
74             if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN): continue
75             #计算误差估计
76             newS = errType(mat0) + errType(mat1)
77             #如果误差比目前的最小, 则更新特征索引值和特征值

```

```

73         #如果误差估计更小,则更新特征索引和特征值
74         if newS < bestS:
75             bestIndex = featIndex
76             bestValue = splitVal
77             bestS = newS
78     #如果误差减少不大则退出
79     if (S - bestS) < tolS:
80         return None, leafType(dataSet)
81     #根据最佳的切分特征和特征值切分数据集
82     mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
83     #如果切分出的数据集很小则退出
84     if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN):
85         return None, leafType(dataSet)
86     #返回最佳切分特征和特征值
87     return bestIndex, bestValue
88
89 # 函数说明: 树构造函数
90 """
91 Parameters:
92     dataSet - 数据集
93     leafType - 建立叶结点的函数
94     errType - 误差计算函数
95     ops - 包含树构建所有其他参数的元组
96 """
97 def createTree(dataSet, leafType = regLeaf, errType = regErr, ops = (1, 4)):
98     #选择最佳切分特征和特征值
99     feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
100     #如果没有特征,则返回特征值
101     if feat == None: return val
102     #回归树
103     retTree = {}
104     retTree['spInd'] = feat
105     retTree['spVal'] = val
106     #分成左数据集和右数据集
107     lSet, rSet = binSplitDataSet(dataSet, feat, val)
108     #创建左子树和右子树
109     retTree['left'] = createTree(lSet, leafType, errType, ops)
110     retTree['right'] = createTree(rSet, leafType, errType, ops)
111     return retTree
112
113 if __name__ == '__main__':
114     myDat = loadDataSet('ex0.txt')
115     myMat = np.mat(myDat)
116     print(createTree(myMat))

```

```

{'spInd': 1, 'spVal': 0.39435, 'left': {'spInd': 1, 'spVal': 0.582002, 'left': {'spInd': 1, 'spVal': 0.797583, 'left': 3.9871631999999999, 'right': 2.9836209534883724}, 'right': 1.980035071428571}, 'right': {'spInd': 1, 'spVal': 0.197834, 'left': 1.0289583666666666, 'right': -0.023838155555555553}}

```

可以看到，该数的结构中包含5个叶结点。

5、树剪枝

一棵树，如果节点过多，表明该模型可能对数据进行了 **过拟合**。那么，如何判断是否发生了过拟合？

通过降低决策树的复杂度来避免过拟合的过程称为 **剪枝 (pruning)**。

- 在函数 `chooseBestSplit()` 中的提前终止条件，实际上是在进行一种所谓的 **预剪枝 (prepruning)** 操作。
- 另一种形式的剪枝需要使用测试集和训练集，称作 **后剪枝 (postpruning)**。

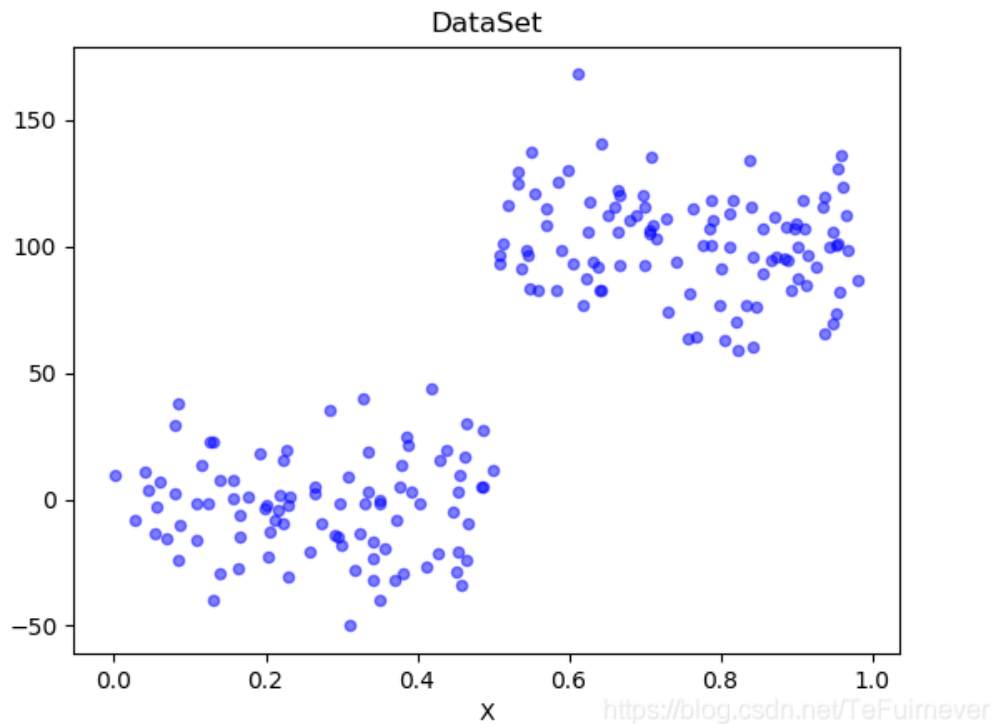
1) 预剪枝

上节两个简单实验的结果还是令人满意的，但背后存在一些问题。树构建算法其实对输入的参数 `tolS` 和 `tolN` 非常敏感，如果使用其他值将不太容易达到这么好的效果。

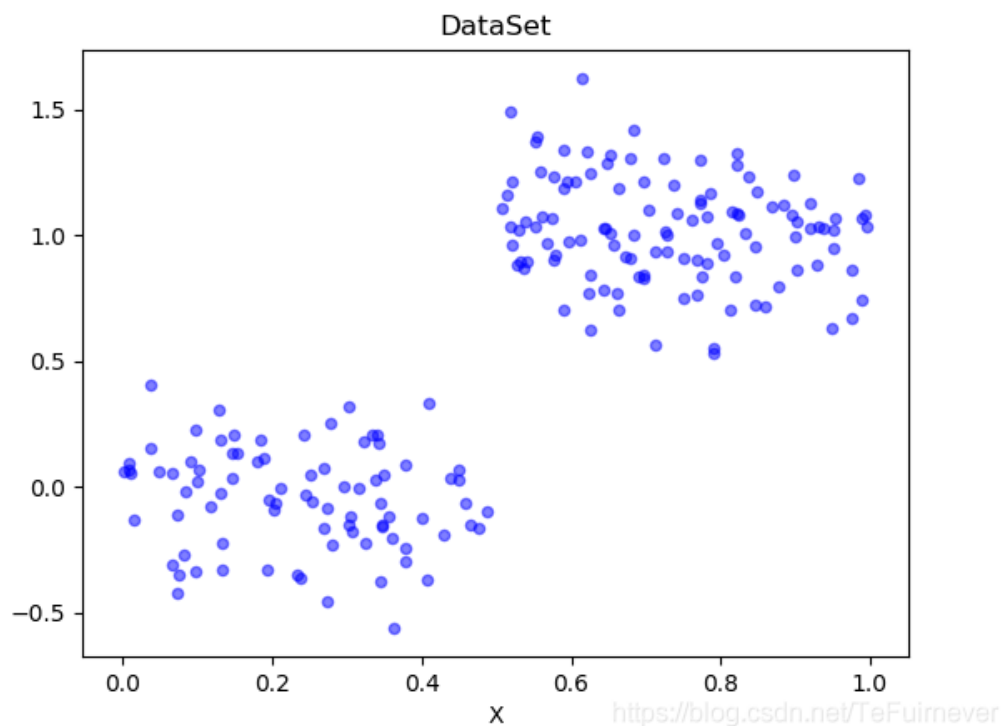
```

1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  # 函数说明: 加载数据
5  """
6  Parameters:
7      fileName - 文件名
8  """
9  def loadDataSet(fileName):
10
11      dataMat = []
12      fr = open(fileName)
13      for line in fr.readlines():
14          curLine = line.strip().split('\t')
15          fltLine = list(map(float, curLine))           # 转化为float类型
16          dataMat.append(fltLine)
17      return dataMat
18
19  # 函数说明: 绘制数据集
20  """
21  Parameters:
22      filename - 文件名
23  """
24  def plotDataSet(filename):
25      dataMat = loadDataSet(filename)                # 加载数据集
26      n = len(dataMat)                                # 数据个数
27      xcord = []; ycord = []                          # 样本点
28      for i in range(n):
29          xcord.append(dataMat[i][0]); ycord.append(dataMat[i][1]) # 样本点
30      fig = plt.figure()
31      ax = fig.add_subplot(111)                        # 添加subplot
32      ax.scatter(xcord, ycord, s = 20, c = 'blue', alpha = .5) # 绘制样本点
33      plt.title('DataSet')                             # 绘制title
34      plt.xlabel('X')
35      plt.show()
36
37  if __name__ == '__main__':
38      filename = 'ex2.txt'
39      plotDataSet(filename)
40

```



可以看到，对于这个数据集与第一个数据集很相似，但是区别在于y的数量级差100倍。下图是第一个数据集的结果：



可以看得出数据分布相似，因此构建出的树应该也是只有两个叶结点。但是使用默认 `tolS` 和 `tolN` 参数创建树，会发现运行结果并不是预想的那样。

```
1 import numpy as np
2
3 # 函数说明: 加载数据
4 """
5 Parameters:
6     fileName - 文件名
7 """
```

```

7  ....
8  def loadDataSet(fileName):
9      dataMat = []
10     fr = open(fileName)
11     for line in fr.readlines():
12         curLine = line.strip().split('\t')
13         fltLine = list(map(float, curLine)) # 转化为float类型
14         dataMat.append(fltLine)
15     return dataMat
16
17 # 函数说明: 根据特征切分数据集
18 """
19 Parameters:
20     dataSet - 数据集
21     feature - 带切分的特征
22     value - 该特征的值
23 """
24 def binSplitDataSet(dataSet, feature, value):
25     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0], :]
26     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0], :]
27     return mat0, mat1
28
29 # 函数说明: 生成叶结点
30 """
31 Parameters:
32     dataSet - 数据集
33 """
34 def regLeaf(dataSet):
35     return np.mean(dataSet[:, -1])
36
37 # 函数说明: 误差估计函数
38 """
39 Parameters:
40     dataSet - 数据集
41 """
42 def regErr(dataSet):
43     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
44
45 # 函数说明: 找到数据的最佳二元切分方式函数
46 """
47 Parameters:
48     dataSet - 数据集
49     leafType - 生成叶结点
50     regErr - 误差估计函数
51     ops - 用户定义的参数构成的元组
52 """
53 def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
54     import types
55     # tolS 允许的误差下降值, tolN 切分的最少样本数
56     tolS = ops[0];
57     tolN = ops[1]
58     # 如果当前所有值相等, 则退出。(根据set的特性)
59     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
60         return None, leafType(dataSet)
61     # 统计数据集的行m和列n
62     m, n = np.shape(dataSet)
63     # 默认最后一个特征为最佳切分特征, 计算其误差估计
64     S = errType(dataSet)
65     # 分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
66     bestS = float('inf');
67     bestIndex = 0;
68     ....

```

```

64     bestValue = 0
65     # 遍历所有特征列
66     for featIndex in range(n - 1):
67         # 遍历所有特征值
68         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
69             # 根据特征和特征值切分数据集
70             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
71             # 如果数据少于tolN, 则退出
72             if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN): continue
73             # 计算误差估计
74             newS = errType(mat0) + errType(mat1)
75             # 如果误差估计更小, 则更新特征索引值和特征值
76             if newS < bestS:
77                 bestIndex = featIndex
78                 bestValue = splitVal
79                 bestS = newS
80         # 如果误差减少不大则退出
81         if (S - bestS) < tolS:
82             return None, leafType(dataSet)
83         # 根据最佳的切分特征和特征值切分数据集
84         mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
85         # 如果切分出的数据集很小则退出
86         if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN):
87             return None, leafType(dataSet)
88         # 返回最佳切分特征和特征值
89         return bestIndex, bestValue
90
91 # 函数说明: 树构造函数
92 """
93 Parameters:
94     dataSet - 数据集
95     leafType - 建立叶结点的函数
96     errType - 误差计算函数
97     ops - 包含树构建所有其他参数的元组
98 """
99
100 def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
101     # 选择最佳切分特征和特征值
102     feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
103     # 如果没有特征, 则返回特征值
104     if feat == None: return val
105     # 回归树
106     retTree = {}
107     retTree['spInd'] = feat
108     retTree['spVal'] = val
109     # 分成左数据集和右数据集
110     lSet, rSet = binSplitDataSet(dataSet, feat, val)
111     # 创建左子树和右子树
112     retTree['left'] = createTree(lSet, leafType, errType, ops)
113     retTree['right'] = createTree(rSet, leafType, errType, ops)
114     return retTree
115
116
117 if __name__ == '__main__':
118     myDat = loadDataSet('ex2.txt')
119     myMat = np.mat(myDat)
120     print(createTree(myMat))

```



```
{'spInd': 0, 'spVal': 0.499171, 'left': {'spInd': 0, 'spVal': 0.729397, 'left': {'spInd': 0, 'spVal': 0.952833, 'left': {'spInd': 0, 'spVal': 0.958512, 'left': 105.24862350000001, 'right': 112.42895575000001}, 'right': {'spInd': 0, 'spVal': 0.759504, 'left': {'spInd': 0, 'spVal': 0.790312, 'left': {'spInd': 0, 'spVal': 0.833026, 'left': {'spInd': 0, 'spVal': 0.944221, 'left': 87.31038750000004, 'right': {'spInd': 0, 'spVal': 0.85497, 'left': {'spInd': 0, 'spVal': 0.910975, 'left': 96.45286699999998, 'right': {'spInd': 0, 'spVal': 0.892999, 'left': 104.82540899999999, 'right': {'spInd': 0, 'spVal': 0.872883, 'left': 95.18179299999999, 'right': 102.25234449999999}}}, 'right': 95.27584316666661}}, 'right': {'spInd': 0, 'spVal': 0.811602, 'left': 81.11015199999999, 'right': 88.78449880000009}}, 'right': 102.35780185714285}, 'right': 78.08564325000004}}, 'right': {'spInd': 0, 'spVal': 0.640515, 'left': {'spInd': 0, 'spVal': 0.666452, 'left': {'spInd': 0, 'spVal': 0.706961, 'left': 114.554706, 'right': {'spInd': 0, 'spVal': 0.698472, 'left': 104.82495374999999, 'right': 108.92921799999999}}, 'right': 114.15162428571431}, 'right': {'spInd': 0, 'spVal': 0.613004, 'left': 93.673449714285724, 'right': {'spInd': 0, 'spVal': 0.582311, 'left': 123.2101316, 'right': {'spInd': 0, 'spVal': 0.553797, 'left': 97.20018024999998, 'right': {'spInd': 0, 'spVal': 0.51915, 'left': {'spInd': 0, 'spVal': 0.543843, 'left': 109.38961049999999, 'right': 110.979946}, 'right': 101.73699325000001}}}}}, 'right': {'spInd': 0, 'spVal': 0.457563, 'left': {'spInd': 0, 'spVal': 0.467383, 'left': 12.50675925, 'right': 3.433133000000007}, 'right': {'spInd': 0, 'spVal': 0.126833, 'left': {'spInd': 0, 'spVal': 0.373501, 'left': {'spInd': 0, 'spVal': 0.437652, 'left': -12.558604833333334, 'right': {'spInd': 0, 'spVal': 0.412516, 'left': 14.38417875, 'right': {'spInd': 0, 'spVal': 0.385021, 'left': -0.8923554999999995, 'right': 3.658477250000016}}}, 'right': {'spInd': 0, 'spVal': 0.335182, 'left': {'spInd': 0, 'spVal': 0.350725, 'left': -15.085111749999999, 'right': -22.693879600000002}, 'right': {'spInd': 0, 'spVal': 0.324274, 'left': 15.059290750000001, 'right': {'spInd': 0, 'spVal': 0.297107, 'left': -19.994155200000002, 'right': {'spInd': 0, 'spVal': 0.166765, 'left': {'spInd': 0, 'spVal': 0.202161, 'left': {'spInd': 0, 'spVal': 0.217214, 'left': {'spInd': 0, 'spVal': 0.228473, 'left': {'spInd': 0, 'spVal': 0.25807, 'left': 0.40377471428571476, 'right': -13.070501}, 'right': 6.770429}, 'right': -11.822278500000001}, 'right': 3.4496025000000001}, 'right': {'spInd': 0, 'spVal': 0.156067, 'left': -12.107972500000001, 'right': -6.247900000000013}}}}}, 'right': {'spInd': 0, 'spVal': 0.084661, 'left': 6.5098432857142843, 'right': {'spInd': 0, 'spVal': 0.044737, 'left': -2.5443927142857148, 'right': 4.0916259999999998}}}}}
```

可以看到，构建出的树有很多叶结点。产生这个现象的原因在于，停止条件 `tolS` 对误差的数量级十分敏感。

如果在选项中花费时间并对上述误差容忍度取平均值，或许也能得到仅有两个叶结点组成的树：

```
1 | if __name__ == '__main__':
2 |     myDat = loadDataSet('ex2.txt')
3 |     myMat = np.mat(myDat)
4 |     print(createTree(myMat, ops=(10000, 4)))
```

```
{'spInd': 0, 'spVal': 0.499171, 'left': 101.35815937735848, 'right': -2.6377193297872341}
```

可以看到，将参数 `tolS` 修改为10000后，构建的树就是只有两个叶结点。然而通过不断修改停止条件来得到合理结果并不是很好的办法。事实上，我们常常甚至不确定到底需要寻找什么样的结果。因为对于一个很多维度的数据集，你也不知道构建的树需要多少个叶结点。可见，预剪枝有很大的局限性。

2) 后剪枝

使用后剪枝方法需要将数据集分成测试集和训练集。

- 首先指定参数，使得构建出的树足够大、足够复杂，便于剪枝。
- 接下来从上而下找到叶节点，用测试集来判断将这些叶节点合并是否能降低测试误差，如果是的话就合并。

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  # 函数说明: 加载数据
5  """
6  Parameters:
7      fileName - 文件名
8  """
9  def loadDataSet(fileName):
10     dataMat = []
11     fr = open(fileName)
12     for line in fr.readlines():
13         curLine = line.strip().split('\t')
14         fltLine = list(map(float, curLine))           # 转化为float类型
15         dataMat.append(fltLine)
16     return dataMat
17
18 # 函数说明: 绘制数据集
19 """
20 Parameters:
21     filename - 文件名
22 """
23 def plotDataSet(filename):
24     dataMat = loadDataSet(filename)                # 加载数据集
25     n = len(dataMat)                                # 数据个数
26     xcord = []; ycord = []                          # 样本点
27     for i in range(n):
28         xcord.append(dataMat[i][0]); ycord.append(dataMat[i][1]) # 样本点
29     fig = plt.figure()
30     ax = fig.add_subplot(111)                        # 添加subplot
31     ax.scatter(xcord, ycord, s = 20, c = 'blue', alpha = .5) # 绘制样本点
32     plt.title('DataSet')                             # 绘制title
33     plt.xlabel('X')
34     plt.show()
35
36 # 函数说明: 根据特征切分数据集
37 """
38 Parameters:
39     dataSet - 数据集
40     feature - 带切分的特征
41     value - 该特征的值
42 """
43 def binSplitDataSet(dataSet, feature, value):
44     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0],:]
45     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0],:]
46     return mat0, mat1
47
48 # 函数说明: 生成叶结点
49 """
50 Parameters:
51     dataSet - 数据集
52 """
53 def regLeaf(dataSet):
54     return np.mean(dataSet[:, -1])
55
56 # 函数说明: 误差估计函数
57 """
58 Parameters:
59     dataSet - 数据集
60 """

```

```

57 def regErr(dataSet):
58     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
59
60 # 函数说明: 找到数据的最佳二元切分方式函数
61 """
62 Parameters:
63     dataSet - 数据集合
64     leafType - 生成叶结点
65     regErr - 误差估计函数
66     ops - 用户定义参数构成的元组
67 """
68 def chooseBestSplit(dataSet, leafType = regLeaf, errType = regErr, ops = (1, 4)):
69     import types
70     # toLS 允许的误差下降值, toLN 切分的最少样本数
71     tolS = ops[0]; tolN = ops[1]
72     # 如果当前所有值相等, 则退出。(根据set的特性)
73     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
74         return None, leafType(dataSet)
75     # 统计数据集合的行m和列n
76     m, n = np.shape(dataSet)
77     # 默认最后一个特征为最佳切分特征, 计算其误差估计
78     S = errType(dataSet)
79     # 分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
80     bestS = float('inf'); bestIndex = 0; bestValue = 0
81     # 遍历所有特征列
82     for featIndex in range(n - 1):
83         # 遍历所有特征值
84         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
85             # 根据特征和特征值切分数据集
86             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
87             # 如果数据少于tolN, 则退出
88             if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN): continue
89             # 计算误差估计
90             newS = errType(mat0) + errType(mat1)
91             # 如果误差估计更小, 则更新特征索引值和特征值
92             if newS < bestS:
93                 bestIndex = featIndex
94                 bestValue = splitVal
95                 bestS = newS
96     # 如果误差减少不大则退出
97     if (S - bestS) < tolS:
98         return None, leafType(dataSet)
99     # 根据最佳的切分特征和特征值切分数据集
100     mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
101     # 如果切分出的数据集很小则退出
102     if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN):
103         return None, leafType(dataSet)
104     # 返回最佳切分特征和特征值
105     return bestIndex, bestValue
106
107 # 函数说明: 树构造函数
108 """
109 Parameters:
110     dataSet - 数据集合
111     leafType - 建立叶结点的函数
112     errType - 误差计算函数
113     ops - 包含树构建所有其他参数的元组
114 """
115 def createTree(dataSet, leafType = regLeaf, errType = regErr, ops = (1, 4)):

```

```

114     #选择最佳切分特征和特征值
115     feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
116     #r如果没有特征,则返回特征值
117     if feat == None: return val
118     #回归树
119     retTree = {}
120     retTree['spInd'] = feat
121     retTree['spVal'] = val
122     #分成左数据集和右数据集
123     lSet, rSet = binSplitDataSet(dataSet, feat, val)
124     #创建左子树和右子树
125     retTree['left'] = createTree(lSet, leafType, errType, ops)
126     retTree['right'] = createTree(rSet, leafType, errType, ops)
127     return retTree
128
129 # 函数说明: 判断测试输入变量是否是一棵树
130 """
131 Parameters:
132     obj - 测试对象
133 """
134 def isTree(obj):
135     import types
136     return (type(obj).__name__ == 'dict')
137
138 # 函数说明: 对树进行塌陷处理(即返回树平均值)
139 """
140 Parameters:
141     tree - 树
142 """
143 def getMean(tree):
144     if isTree(tree['right']): tree['right'] = getMean(tree['right'])
145     if isTree(tree['left']): tree['left'] = getMean(tree['left'])
146     return (tree['left'] + tree['right']) / 2.0
147
148 # 函数说明: 后剪枝
149 """
150 Parameters:
151     tree - 树
152     test - 测试集
153 """
154 def prune(tree, testData):
155     #如果测试集为空,则对树进行塌陷处理
156     if np.shape(testData)[0] == 0: return getMean(tree)
157     #如果有左子树或者右子树,则切分数据集
158     if (isTree(tree['right']) or isTree(tree['left'])):
159         lSet, rSet = binSplitDataSet(testData, tree['spInd'], tree['spVal'])
160     #处理左子树(剪枝)
161     if isTree(tree['left']): tree['left'] = prune(tree['left'], lSet)
162     #处理右子树(剪枝)
163     if isTree(tree['right']): tree['right'] = prune(tree['right'], rSet)
164     #如果当前结点的左右结点为叶结点
165     if not isTree(tree['left']) and not isTree(tree['right']):
166         lSet, rSet = binSplitDataSet(testData, tree['spInd'], tree['spVal'])
167         #计算没有合并的误差
168         errorNoMerge = np.sum(np.power(lSet[:, -1] - tree['left'], 2)) + np.sum(np.power(rSet[:, -1] -
169         #计算合并的均值
170         treeMean = (tree['left'] + tree['right']) / 2.0
171         #计算合并的误差
172         errorMerge = np.sum(np.power(testData[:, -1] - treeMean, 2))

```

```

171         #如果合并的误差小于没有合并的误差,则合并
172         if errorMerge < errorNoMerge:
173             return treeMean
174         else: return tree
175     else: return tree
176
177 if __name__ == '__main__':
178     train_filename = 'ex2.txt'
179     train_Data = loadDataSet(train_filename)
180     train_Mat = np.mat(train_Data)
181     tree = createTree(train_Mat)
182     print(tree)
183     test_filename = 'ex2test.txt'
184     test_Data = loadDataSet(test_filename)
185     test_Mat = np.mat(test_Data)
186     print(prune(tree, test_Mat))
187
188

```

```

C:\Users\Administrat\Anaconda3\python.exe E:/机器学习实战/机器学习/机器学习实战/Ch09/11111111.py {'spInd': 0,
'spVal': 0.499171, 'left': {'spInd': 0, 'spVal': 0.729397, 'left': {'spInd': 0, 'spVal': 0.952833, 'left':
{'spInd': 0, 'spVal': 0.958512, 'left': 105.24862350000001, 'right': 112.42895575000001}, 'right': {'spInd':
0, 'spVal': 0.759504, 'left': {'spInd': 0, 'spVal': 0.790312, 'left': {'spInd': 0, 'spVal': 0.833026, 'left':
{'spInd': 0, 'spVal': 0.944221, 'left': 87.310387500000004, 'right': {'spInd': 0, 'spVal': 0.85497, 'left':
{'spInd': 0, 'spVal': 0.910975, 'left': 96.452866999999998, 'right': {'spInd': 0, 'spVal': 0.892999, 'left':
104.82540899999999, 'right': {'spInd': 0, 'spVal': 0.872883, 'left': 95.18179299999999, 'right':
102.25234449999999}}}, 'right': 95.275843166666661}}, 'right': {'spInd': 0, 'spVal': 0.811602, 'left':
81.11015199999999, 'right': 88.784498800000009}}, 'right': 102.35780185714285}, 'right':
78.085643250000004}}, 'right': {'spInd': 0, 'spVal': 0.640515, 'left': {'spInd': 0, 'spVal': 0.666452, 'left':
{'spInd': 0, 'spVal': 0.706961, 'left': 114.554706, 'right': {'spInd': 0, 'spVal': 0.698472, 'left':
104.82495374999999, 'right': 108.92921799999999}}, 'right': 114.15162428571431}, 'right': {'spInd': 0,
'spVal': 0.613004, 'left': 93.673449714285724, 'right': {'spInd': 0, 'spVal': 0.582311, 'left': 123.2101316,
'right': {'spInd': 0, 'spVal': 0.553797, 'left': 97.20018024999998, 'right': {'spInd': 0, 'spVal': 0.51915,
'left': {'spInd': 0, 'spVal': 0.543843, 'left': 109.38961049999999, 'right': 110.979946}, 'right':
101.73699325000001}}}}}, 'right': {'spInd': 0, 'spVal': 0.457563, 'left': {'spInd': 0, 'spVal': 0.467383,
'left': 12.50675925, 'right': 3.4331330000000007}, 'right': {'spInd': 0, 'spVal': 0.126833, 'left': {'spInd':
0, 'spVal': 0.373501, 'left': {'spInd': 0, 'spVal': 0.437652, 'left': -12.558604833333334, 'right': {'spInd':
0, 'spVal': 0.412516, 'left': 14.38417875, 'right': {'spInd': 0, 'spVal': 0.385021, 'left':
-0.89235549999999952, 'right': 3.6584772500000016}}}, 'right': {'spInd': 0, 'spVal': 0.335182, 'left':
{'spInd': 0, 'spVal': 0.350725, 'left': -15.085111749999999, 'right': -22.693879600000002}, 'right': {'spInd':
0, 'spVal': 0.324274, 'left': 15.059290750000001, 'right': {'spInd': 0, 'spVal': 0.297107, 'left':
-19.994155200000002, 'right': {'spInd': 0, 'spVal': 0.166765, 'left': {'spInd': 0, 'spVal': 0.202161, 'left':
{'spInd': 0, 'spVal': 0.217214, 'left': {'spInd': 0, 'spVal': 0.228473, 'left': {'spInd': 0, 'spVal': 0.25807,
'left': 0.40377471428571476, 'right': -13.070501}, 'right': 6.770429}, 'right': -11.822278500000001}, 'right':
3.4496025000000001}, 'right': {'spInd': 0, 'spVal': 0.156067, 'left': -12.107972500000001, 'right':
-6.2479000000000013}}}}}, 'right': {'spInd': 0, 'spVal': 0.084661, 'left': 6.5098432857142843, 'right':
{'spInd': 0, 'spVal': 0.044737, 'left': -2.5443927142857148, 'right': 4.0916259999999998}}}}}}

```

```

{'spInd': 0, 'spVal': 0.499171, 'left': {'spInd': 0, 'spVal': 0.729397, 'left': {'spInd': 0, 'spVal':
0.952833, 'left': {'spInd': 0, 'spVal': 0.958512, 'left': 105.24862350000001, 'right': 112.42895575000001},
'right': {'spInd': 0, 'spVal': 0.759504, 'left': {'spInd': 0, 'spVal': 0.790312, 'left': {'spInd': 0, 'spVal':
0.833026, 'left': {'spInd': 0, 'spVal': 0.944221, 'left': 87.310387500000004, 'right': {'spInd': 0, 'spVal':
0.85497, 'left': {'spInd': 0, 'spVal': 0.910975, 'left': 96.452866999999998, 'right': {'spInd': 0, 'spVal':
0.892999, 'left': 104.82540899999999, 'right': {'spInd': 0, 'spVal': 0.872883, 'left': 95.18179299999999,
'right': 102.25234449999999}}}, 'right': 95.275843166666661}}, 'right': {'spInd': 0, 'spVal': 0.811602,
'left': 81.11015199999999, 'right': 88.784498800000009}}, 'right': 102.35780185714285}, 'right':

```

```
78.085643250000004}}, 'right': {'spInd': 0, 'spVal': 0.640515, 'left': {'spInd': 0, 'spVal': 0.666452, 'left':
{'spInd': 0, 'spVal': 0.706961, 'left': 114.554706, 'right': 106.87708587499999}, 'right':
114.15162428571431}, 'right': {'spInd': 0, 'spVal': 0.613004, 'left': 93.673449714285724, 'right': {'spInd':
0, 'spVal': 0.582311, 'left': 123.2101316, 'right': 101.580533}}}}, 'right': {'spInd': 0, 'spVal': 0.457563,
'left': 7.9699461249999999, 'right': {'spInd': 0, 'spVal': 0.126833, 'left': {'spInd': 0, 'spVal': 0.373501,
'left': {'spInd': 0, 'spVal': 0.437652, 'left': -12.558604833333334, 'right': {'spInd': 0, 'spVal': 0.412516,
'left': 14.38417875, 'right': 1.3830608750000011}, 'right': {'spInd': 0, 'spVal': 0.335182, 'left': {'spInd':
0, 'spVal': 0.350725, 'left': -15.085111749999999, 'right': -22.693879600000002}, 'right': {'spInd': 0,
'spVal': 0.324274, 'left': 15.059290750000001, 'right': {'spInd': 0, 'spVal': 0.297107, 'left':
-19.994155200000002, 'right': {'spInd': 0, 'spVal': 0.166765, 'left': {'spInd': 0, 'spVal': 0.202161, 'left':
-5.801872785714286, 'right': 3.4496025000000001}, 'right': {'spInd': 0, 'spVal': 0.156067, 'left':
-12.107972500000001, 'right': -6.2479000000000013}}}}}}, 'right': {'spInd': 0, 'spVal': 0.084661, 'left':
6.5098432857142843, 'right': {'spInd': 0, 'spVal': 0.044737, 'left': -2.5443927142857148, 'right':
4.0916259999999998}}}}}
```

现在使用 `ex2.txt` 训练回归树，然后利用 `ex2test.txt` 对回归树进行剪枝。可以看到，树的大量结点已经被剪枝掉了，但没有像预期的那样剪枝成两部分，这说明后剪枝可能不如预剪枝有效。

一般地，为了寻求最佳模型可以同时使用两种剪枝技术。

6、模型树

用树来对数据建模，除了把叶节点简单地设定为常数值之外，还有一种方法是把叶节点设定为分段线性函数，这里所谓的分段线性（piecewise linear）是指模型由多个线性片段组成。使用两条直线拟合是否比使用一组常数来建模好呢？答案显而易见。可以把数据集里的一部分数据（0.0~0.3）以某个线性模型建模，而另一部分数据（0.3~1.0）则以另一个线性模型建模，因此说采用了所谓的分段线性模型。

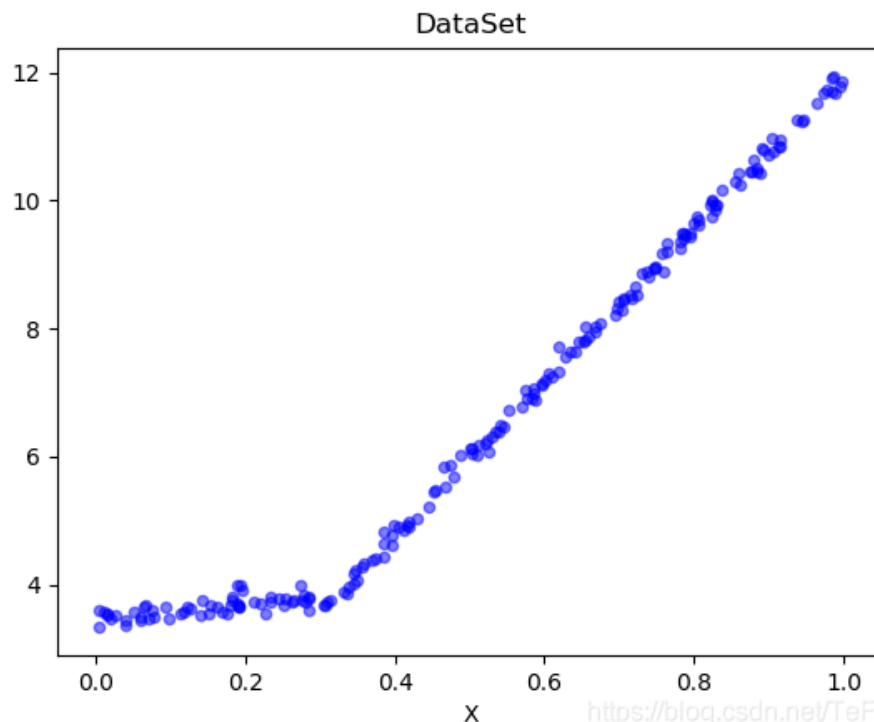
决策树相比于其他机器学习算法的优势之一在于结果更易理解。很显然，两条直线比很多节点组成一棵大树更容易解释。模型树的可解释性是它优于回归树的特点之一。另外，模型树也具有更高的预测准确度。

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  # 函数说明: 加载数据
5  """
6  Parameters:
7      fileName - 文件名
8  """
9  def loadDataSet(fileName):
10
11      dataMat = []
12      fr = open(fileName)
13      for line in fr.readlines():
14          curLine = line.strip().split('\t')
15          fltLine = list(map(float, curLine))           # 转化为float类型
16          dataMat.append(fltLine)
17      return dataMat
18
19  # 函数说明: 绘制数据集
20  """
21  Parameters:
22      filename - 文件名
23  """
24  def plotDataSet(filename):
25
26      dataMat = loadDataSet(filename)                # 加载数据集
27      n = len(dataMat)                                # 数据个数
28      xcord = []; ycord = []                          # 样本点
```

```

27     for i in range(n):
28         xcord.append(dataMat[i][0]); ycord.append(dataMat[i][1]) #样本点
29     fig = plt.figure()
30     ax = fig.add_subplot(111) #添加subplot
31     ax.scatter(xcord, ycord, s = 20, c = 'blue', alpha = .5) #绘制样本点
32     plt.title('DataSet') #绘制title
33     plt.xlabel('X')
34     plt.show()
35
36
37 if __name__ == '__main__':
38     filename = 'exp2.txt'
39     plotDataSet(filename)
40

```



<https://blog.csdn.net/TeFuirnever>

```

1  import numpy as np
2
3  # 函数说明: 加载数据
4  """
5  Parameters:
6      fileName - 文件名
7  """
8  def loadDataSet(fileName):
9      dataMat = []
10     fr = open(fileName)
11     for line in fr.readlines():
12         curLine = line.strip().split('\t')
13         fltLine = list(map(float, curLine)) # 转化为float类型
14         dataMat.append(fltLine)
15     return dataMat
16
17 # 函数说明: 根据特征切分数据集
18 """
19 Parameters:
20     dataSet - 数据集
21     feature - 带切分的特征

```



```

20     value - 该特征的值
21 """
22 def binSplitDataSet(dataSet, feature, value):
23     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0], :]
24     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0], :]
25     return mat0, mat1
26
27 # 函数说明: 生成叶结点
28 """
29 Parameters:
30     dataSet - 数据集
31 """
32 def regLeaf(dataSet):
33     return np.mean(dataSet[:, -1])
34
35 # 函数说明: 误差估计函数
36 """
37 Parameters:
38     dataSet - 数据集
39 """
40 def regErr(dataSet):
41     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
42
43 # 函数说明: 找到数据的最佳二元切分方式函数
44 """
45 Parameters:
46     dataSet - 数据集
47     leafType - 生成叶结点
48     regErr - 误差估计函数
49     ops - 用户定义参数构成的元组
50 """
51 def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
52     import types
53     # tolS允许的误差下降值, tolN切分的最少样本数
54     tolS = ops[0];
55     tolN = ops[1]
56     # 如果当前所有值相等, 则退出。(根据set的特性)
57     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
58         return None, leafType(dataSet)
59     # 统计数据集的行m和列n
60     m, n = np.shape(dataSet)
61     # 默认最后一个特征为最佳切分特征, 计算其误差估计
62     S = errType(dataSet)
63     # 分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
64     bestS = float('inf');
65     bestIndex = 0;
66     bestValue = 0
67     # 遍历所有特征列
68     for featIndex in range(n - 1):
69         # 遍历所有特征值
70         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
71             # 根据特征和特征值切分数据集
72             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
73             # 如果数据少于tolN, 则退出
74             if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN): continue
75             # 计算误差估计
76             newS = errType(mat0) + errType(mat1)
77             # 如果误差估计更小, 则更新特征索引值和特征值
78             if newS < bestS:
79                 bestIndex = featIndex

```



```

77         bestValue = splitVal
78         bestS = newS
79         # 如果误差减少不大则退出
80         if (S - bestS) < tolS:
81             return None, leafType(dataSet)
82         # 根据最佳的切分特征和特征值切分数据集
83         mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
84         # 如果切分出的数据集很小则退出
85         if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN):
86             return None, leafType(dataSet)
87         # 返回最佳切分特征和特征值
88         return bestIndex, bestValue
89
90 # 函数说明: 树构造函数
91 """
92 Parameters:
93     dataSet - 数据集
94     leafType - 建立叶结点的函数
95     errType - 误差计算函数
96     ops - 包含树构建所有其他参数的元组
97 """
98 def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
99     # 选择最佳切分特征和特征值
100     feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
101     # 如果没有特征, 则返回特征值
102     if feat == None: return val
103     # 回归树
104     retTree = {}
105     retTree['spInd'] = feat
106     retTree['spVal'] = val
107     # 分成左数据集和右数据集
108     lSet, rSet = binSplitDataSet(dataSet, feat, val)
109     # 创建左子树和右子树
110     retTree['left'] = createTree(lSet, leafType, errType, ops)
111     retTree['right'] = createTree(rSet, leafType, errType, ops)
112     return retTree
113
114 def linearSolve(dataSet): #helper function used in two places
115     m,n = np.shape(dataSet)
116     X = np.mat(np.ones((m,n))); Y = np.mat(np.ones((m,1)))#create a copy of data with 1 in 0th posti
117     X[:,1:n] = dataSet[:,0:n-1]; Y = dataSet[:, -1]#and strip out Y
118     xTx = X.T*X
119     if np.linalg.det(xTx) == 0.0:
120         raise NameError('This matrix is singular, cannot do inverse,\n
121         try increasing the second value of ops')
122     ws = xTx.I * (X.T * Y)
123     return ws,X,Y
124
125 def modelLeaf(dataSet):#create linear model and return coefficients
126     ws,X,Y = linearSolve(dataSet)
127     return ws
128
129 def modelErr(dataSet):
130     ws,X,Y = linearSolve(dataSet)
131     yHat = X * ws
132     return np.sum(np.power(Y - yHat,2))
133

```

```

134 if __name__ == '__main__':
135     myDat = loadDataSet('exp2.txt')
136     myMat = np.mat(myDat)
137     print(createTree(myMat, modelLeaf, modelErr,(1,10)))
138

```

```

{'spInd': 0, 'spVal': 0.285477, 'left': matrix([[ 1.69855694e-03],
[ 1.19647739e+01]]), 'right': matrix([[ 3.46877936],
[ 1.18521743]])}

```

可以看到，该代码以 0.285477 为界创建了两个模型。

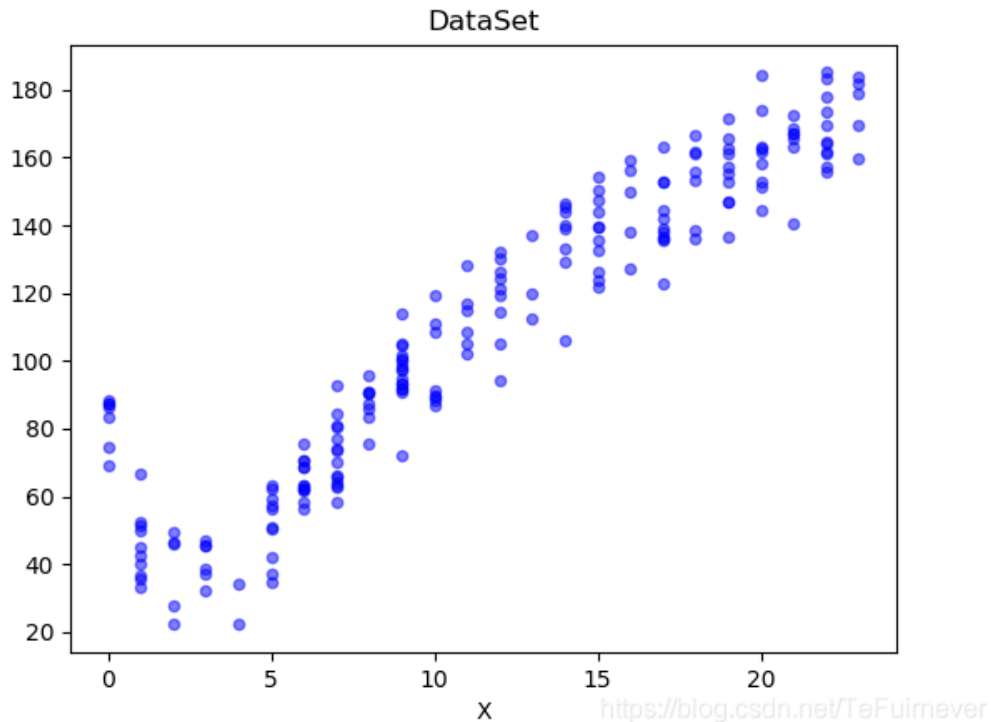
7、示例：树回归与标准回归的比较

介绍了模型树、回归树和一般的回归方法，下面测试一下哪个模型最好。

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  # 函数说明: 加载数据
5  """
6  Parameters:
7      fileName - 文件名
8  """
9  def loadDataSet(fileName):
10
11      dataMat = []
12      fr = open(fileName)
13      for line in fr.readlines():
14          curLine = line.strip().split('\t')
15          fltLine = list(map(float, curLine))           # 转化为float类型
16          dataMat.append(fltLine)
17      return dataMat
18
19  # 函数说明: 绘制数据集
20  """
21  Parameters:
22      filename - 文件名
23  """
24  def plotDataSet(filename):
25      dataMat = loadDataSet(filename)                # 加载数据集
26      n = len(dataMat)                                # 数据个数
27      xcord = []; ycord = []                          # 样本点
28      for i in range(n):
29          xcord.append(dataMat[i][0]); ycord.append(dataMat[i][1]) # 样本点
30      fig = plt.figure()
31      ax = fig.add_subplot(111)                        # 添加subplot
32      ax.scatter(xcord, ycord, s = 20, c = 'blue', alpha = .5) # 绘制样本点
33      plt.title('DataSet')                            # 绘制title
34      plt.xlabel('X')
35      plt.show()
36
37  if __name__ == '__main__':
38      filename = 'bikeSpeedVsIq_train.txt'
39      plotDataSet(filename)

```



上图的数据是从多个骑自行车的人那里收集得到的，图中给出骑自行车的速度和人的智商之间的关系。

接下来，利用该数据创建一棵回归树：

```
1  import matplotlib.pyplot as plt
2  import regTrees
3  import numpy as np
4
5  # 函数说明: 加载数据
6  """
7  Parameters:
8      fileName - 文件名
9  """
10 def loadDataSet(fileName):
11     dataMat = []
12     fr = open(fileName)
13     for line in fr.readlines():
14         curLine = line.strip().split('\t')
15         fltLine = list(map(float, curLine)) # 转化为float类型
16         dataMat.append(fltLine)
17     return dataMat
18
19 # 函数说明: 根据特征切分数据集
20 """
21 Parameters:
22     dataSet - 数据集
23     feature - 带切分的特征
24     value - 该特征的值
25 """
26 def binSplitDataSet(dataSet, feature, value):
27     mat0 = dataSet[np.nonzero(dataSet[:, feature] > value)[0], :]
28     mat1 = dataSet[np.nonzero(dataSet[:, feature] <= value)[0], :]
29     return mat0, mat1
```

```

30 # 函数说明: 生成叶结点
31 """
32 Parameters:
33     dataSet - 数据集合
34 """
35 def regLeaf(dataSet):
36     return np.mean(dataSet[:, -1])
37
38 # 函数说明: 误差估计函数
39 """
40 Parameters:
41     dataSet - 数据集合
42 """
43 def regErr(dataSet):
44     return np.var(dataSet[:, -1]) * np.shape(dataSet)[0]
45
46 # 函数说明: 找到数据的最佳二元切分方式函数
47 """
48 Parameters:
49     dataSet - 数据集合
50     leafType - 生成叶结点
51     regErr - 误差估计函数
52     ops - 用户定义的参数构成的元组
53 """
54 def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
55     import types
56     # tolS允许的误差下降值, tolN切分的最少样本数
57     tolS = ops[0];
58     tolN = ops[1]
59     # 如果当前所有值相等, 则退出。(根据set的特性)
60     if len(set(dataSet[:, -1].T.tolist()[0])) == 1:
61         return None, leafType(dataSet)
62     # 统计数据集合的行m和列n
63     m, n = np.shape(dataSet)
64     # 默认最后一个特征为最佳切分特征, 计算其误差估计
65     S = errType(dataSet)
66     # 分别为最佳误差, 最佳特征切分的索引值, 最佳特征值
67     bestS = float('inf');
68     bestIndex = 0;
69     bestValue = 0
70     # 遍历所有特征列
71     for featIndex in range(n - 1):
72         # 遍历所有特征值
73         for splitVal in set(dataSet[:, featIndex].T.A.tolist()[0]):
74             # 根据特征和特征值切分数据集
75             mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
76             # 如果数据少于tolN, 则退出
77             if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN): continue
78             # 计算误差估计
79             newS = errType(mat0) + errType(mat1)
80             # 如果误差估计更小, 则更新特征索引值和特征值
81             if newS < bestS:
82                 bestIndex = featIndex
83                 bestValue = splitVal
84                 bestS = newS
85     # 如果误差减少不大则退出
86     if (S - bestS) < tolS:
87         return None, leafType(dataSet)
88     # 根据最佳的切分特征和特征值切分数据集

```

```

87     mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
88     # 如果切分出的数据集很小则退出
89     if (np.shape(mat0)[0] < tolN) or (np.shape(mat1)[0] < tolN):
90         return None, leafType(dataSet)
91     # 返回最佳切分特征和特征值
92     return bestIndex, bestValue
93
94 # 函数说明: 树构造函数
95 """
96 Parameters:
97     dataSet - 数据集
98     leafType - 建立叶结点的函数
99     errType - 误差计算函数
100     ops - 包含树构建所有其他参数的元组
101 """
102 def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
103     # 选择最佳切分特征和特征值
104     feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
105     # 如果没有特征, 则返回特征值
106     if feat == None: return val
107     # 回归树
108     retTree = {}
109     retTree['spInd'] = feat
110     retTree['spVal'] = val
111     # 分成左数据集和右数据集
112     lSet, rSet = binSplitDataSet(dataSet, feat, val)
113     # 创建左子树和右子树
114     retTree['left'] = createTree(lSet, leafType, errType, ops)
115     retTree['right'] = createTree(rSet, leafType, errType, ops)
116     return retTree
117
118 def linearSolve(dataSet): #helper function used in two places
119     m,n = np.shape(dataSet)
120     X = np.mat(np.ones((m,n))); Y = np.mat(np.ones((m,1)))#create a copy of data with 1 in 0th posti
121     X[:,1:n] = dataSet[:,0:n-1]; Y = dataSet[:, -1]#and strip out Y
122     xTx = X.T*X
123     if np.linalg.det(xTx) == 0.0:
124         raise NameError('This matrix is singular, cannot do inverse,\n
125         try increasing the second value of ops')
126     ws = xTx.I * (X.T * Y)
127     return ws,X,Y
128
129 def modelLeaf(dataSet):#create linear model and return coefficients
130     ws,X,Y = linearSolve(dataSet)
131     return ws
132
133 def modelErr(dataSet):
134     ws,X,Y = linearSolve(dataSet)
135     yHat = X * ws
136     return np.sum(np.power(Y - yHat,2))
137
138 def isTree(obj):
139     return (type(obj).__name__=='dict')
140
141 def regTreeEval(model, inDat):
142     return float(model)
143
144 def modelTreeEval(model, inDat):

```

```

144     n = np.shape(inDat)[1]
145     X = np.mat(np.ones((1,n+1)))
146     X[:,1:n+1]=inDat
147     return float(X*model)
148
149 def treeForeCast(tree, inData, modelEval=regTreeEval):
150     if not isTree(tree): return modelEval(tree, inData)
151     if inData[tree['spInd']] > tree['spVal']:
152         if isTree(tree['left']): return treeForeCast(tree['left'], inData, modelEval)
153         else: return modelEval(tree['left'], inData)
154     else:
155         if isTree(tree['right']): return treeForeCast(tree['right'], inData, modelEval)
156         else: return modelEval(tree['right'], inData)
157
158 def createForeCast(tree, testData, modelEval=regTreeEval):
159     m=len(testData)
160     yHat = np.mat(np.zeros((m,1)))
161     for i in range(m):
162         yHat[i,0] = treeForeCast(tree, np.mat(testData[i]), modelEval)
163     return yHat
164
165
166 if __name__ == '__main__':
167     trainMat = np.mat(loadDataSet("bikeSpeedVsIq_train.txt"))
168     testMat = np.mat(loadDataSet("bikeSpeedVsIq_test.txt"))
169     myTree = createTree(trainMat, ops=(1, 20))
170     yHat = createForeCast(myTree, testMat[:,0])
171     print(np.corrcoef(yHat, testMat[:,1], rowvar=0)[0, 1])
172

```

0.964085231822

同样地，再创建一棵模型树：

```

1  if __name__ == '__main__':
2      trainMat = np.mat(loadDataSet("bikeSpeedVsIq_train.txt"))
3      testMat = np.mat(loadDataSet("bikeSpeedVsIq_test.txt"))
4      myTree = createTree(trainMat, modelLeaf, modelErr, (1, 20))
5      yHat = createForeCast(myTree, testMat[:,0], modelTreeEval)
6      print(np.corrcoef(yHat, testMat[:,1], rowvar=0)[0, 1])

```

0.976041219138

我们知道， R^2 值越接近1.0越好，所以从上面的结果可以看出，这里模型树的结果比回归树好。

下面再看看标准的线性回归效果如何：

```

1  if __name__ == '__main__':
2      trainMat = np.mat(loadDataSet("bikeSpeedVsIq_train.txt"))
3      testMat = np.mat(loadDataSet("bikeSpeedVsIq_test.txt"))
4      myTree = createTree(trainMat, modelLeaf, modelErr, (1, 20))
5

```

```

6     yHat = createForeCast(myTree, testMat[:,0], modelTreeEval)
7     ws,X,Y = linearSolve(trainMat)
8     for i in range(np.shape(testMat)[0]):
9         yHat[i] = testMat[i,0]*ws[1,0]+ws[0,0]
    print(np.corrcoef(yHat,testMat[:,1],rowvar=0)[0,1])

```

0.943468423567

可以看到，该方法在 R^2 值上的表现上不如上面两种树回归方法。所以，树回归方法在预测复杂数据时会比简单的线性模型更有效。

8、使用 Python 的 Tkinter 库创建 GUI

机器学习提供了一些强大的工具，能从未知数据中抽取有用的信息。因此，能否将这些信息以易于人们理解的方式呈现十分重要。再者，假如人们可以直接与算法和数据交互，将可以比较轻松地进行解释。如果仅仅只是绘制出一幅静态图像，或者只是在Python命令行中输出一些数字，那么对结果做分析和交流将非常困难。如果能让用户不需要任何指令就可以按照他们自己的方式来分析数据，就不需要对数据做出过多解释。其中一个能同时支持数据呈现和用户交互的方式就是构建一个 **图形用户界面 (GUI, Graphical User Interface)**，如图所示。

```

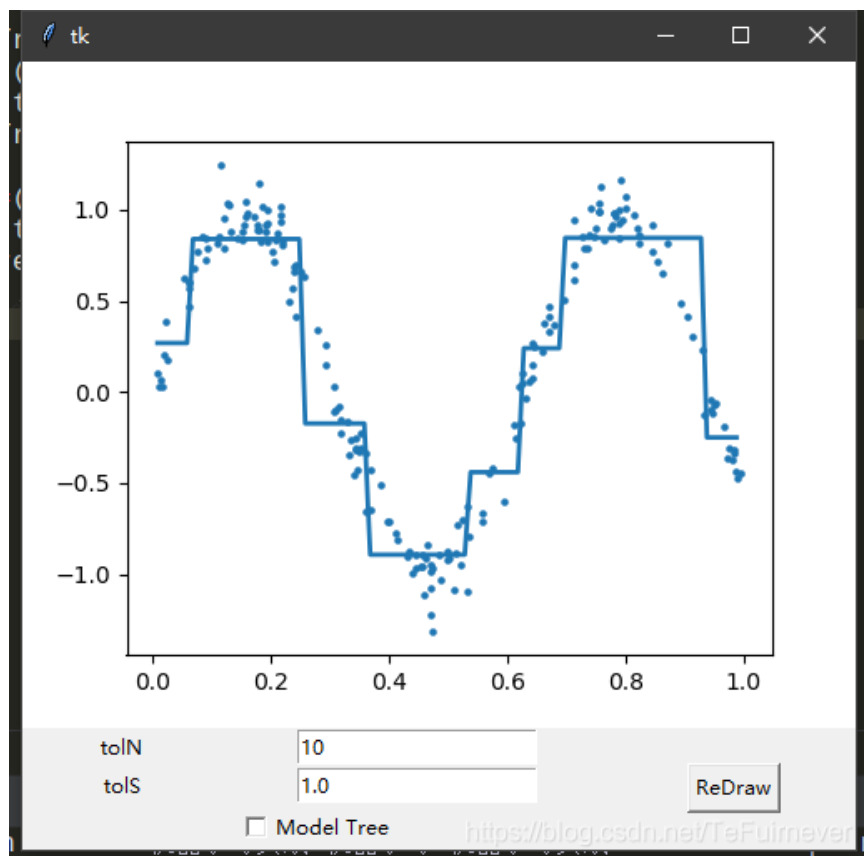
1  from numpy import *
2
3  from tkinter import *
4  import regTrees
5
6  import matplotlib
7  matplotlib.use('TkAgg')
8  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
9  from matplotlib.figure import Figure
10
11 # 绘制树
12 def reDraw(tolS,tolN):
13     reDraw.f.clf()          # clear the figure
14     reDraw.a = reDraw.f.add_subplot(111)
15     if chkBtnVar.get():
16         if tolN < 2: tolN = 2
17         myTree=regTrees.createTree(reDraw.rawDat, regTrees.modelLeaf,\
18                                   regTrees.modelErr, (tolS,tolN))
19         yHat = regTrees.createForeCast(myTree, reDraw.testDat, \
20                                       regTrees.modelTreeEval)
21     else:
22         myTree=regTrees.createTree(reDraw.rawDat, ops=(tolS,tolN))
23         yHat = regTrees.createForeCast(myTree, reDraw.testDat)
24     reDraw.a.scatter(array(reDraw.rawDat[:,0]),array(reDraw.rawDat[:,1]),s= 5) # 离散型散点图
25     reDraw.a.plot(reDraw.testDat, yHat, linewidth=2.0)
26     reDraw.canvas.draw()
27
28 def getInputs():
29     try: tolN = int(tolNentry.get())
30     except:
31         tolN = 10
32         print("enter Integer for tolN")
33         tolNentry.delete(0, END)

```

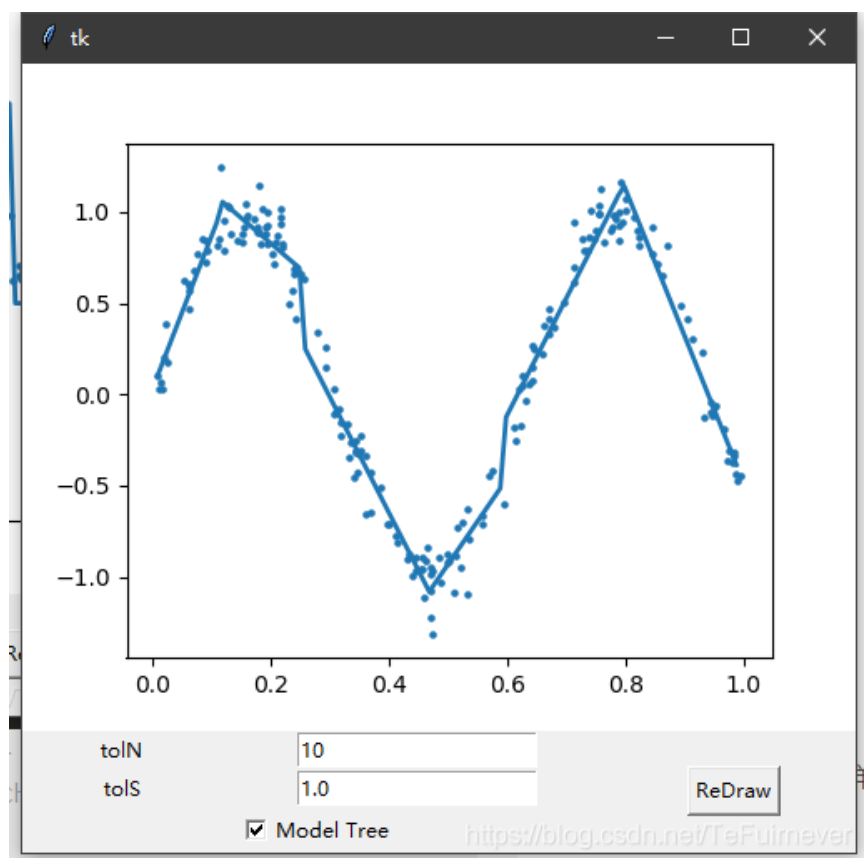
```

34         tolNentry.insert(0, '10')
35     try: tolS = float(tolSentry.get())
36     except:
37         tolS = 1.0
38         print("enter Float for tolS")
39         tolSentry.delete(0, END)
40         tolSentry.insert(0, '1.0')
41     return tolN, tolS
42
43 # 理解用户输入并防止程序崩溃
44 def drawNewTree():
45     tolN, tolS = getInputs() # 从输入框中获取值
46     reDraw(tolS, tolN)      # 生成图
47
48 # Tk类型的根部件
49 root = Tk()
50
51 # 创造画布
52 reDraw.f = Figure(figsize=(5, 4), dpi=100)
53 # 调用Agg, 把Agg呈现在画布上
54 # Agg是一个C++的库, 可以从图像创建光栅图
55 reDraw.canvas = FigureCanvasTkAgg(reDraw.f, master=root)
56 reDraw.canvas.draw()
57 reDraw.canvas.get_tk_widget().grid(row=0, columnspan=3)
58
59 Label(root, text="tolN").grid(row=1, column=0)
60 # 文本输入框1
61 tolNentry = Entry(root)
62 tolNentry.grid(row=1, column=1)
63 tolNentry.insert(0, '10')
64 Label(root, text="tolS").grid(row=2, column=0)
65 # 文本输入框2
66 tolSentry = Entry(root)
67 tolSentry.grid(row=2, column=1)
68 tolSentry.insert(0, '1.0')
69 # 初始化与reDraw()关联的全局变量
70 Button(root, text="ReDraw", command=drawNewTree).grid(row=1, column=2, rowspan=3)
71 # 按钮整数值
72 chkBtnVar = IntVar()
73 # 复选按钮
74 chkBtn = Checkbutton(root, text="Model Tree", variable = chkBtnVar)
75 chkBtn.grid(row=3, column=0, columnspan=2)
76
77 reDraw.rawDat = mat(regTrees.loadDataSet('sine.txt'))
78 reDraw.testDat = arange(min(reDraw.rawDat[:, 0]), max(reDraw.rawDat[:, 0]), 0.01)
79 reDraw(1.0, 10)
80
81 root.mainloop()

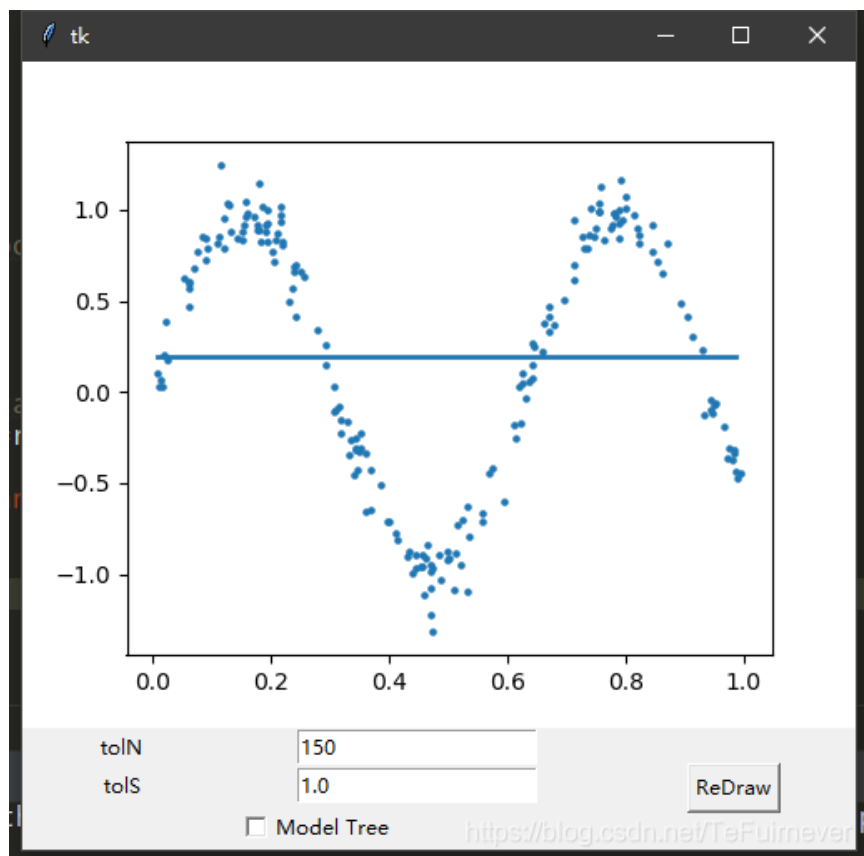
```

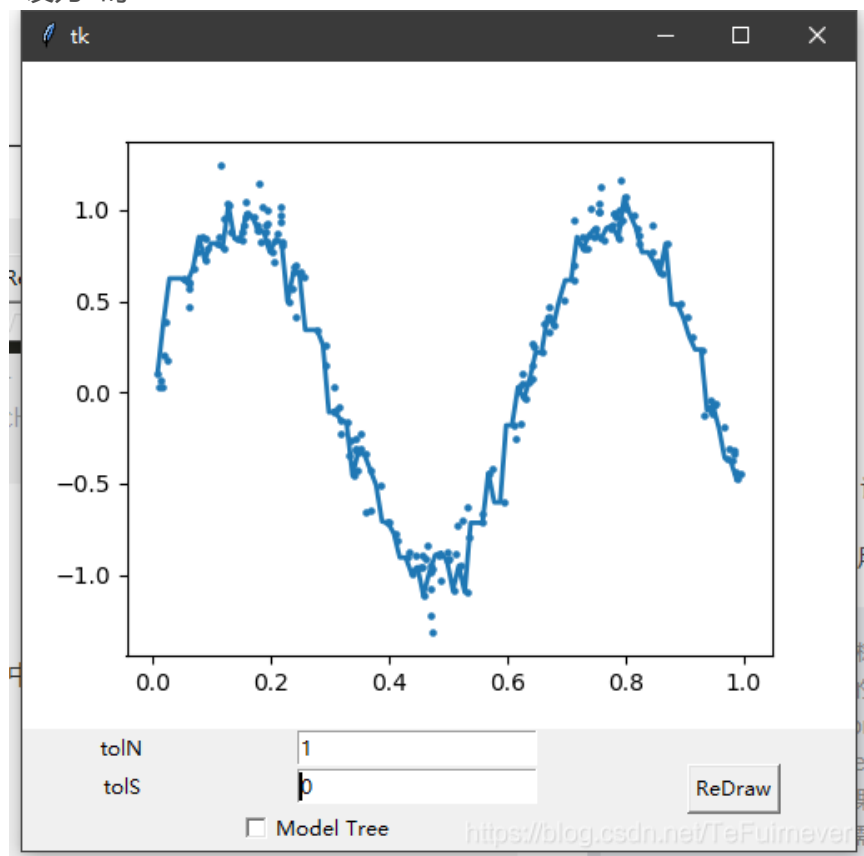
设置 model tree 时



整个数据集包含200个样本，将 `tolN` 设为150，`tolS` 不变时：



将 `tolN` 设为1, `tolS` 设为0时:



默认的treeExplore图形用户界面, 该界面同时显示了输入数据和一个回归树模型, 其中的参数 `tolN=10`, `tolS=1.0`。

示例: 利用GUI对回归树调优

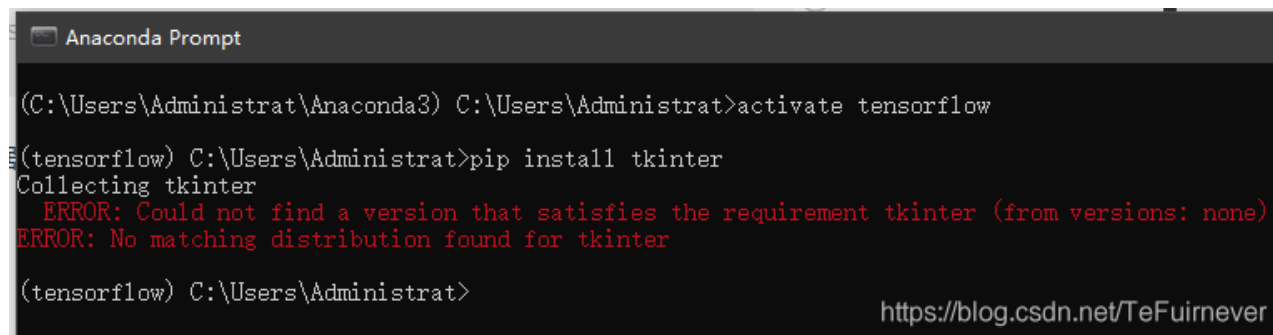
(1) 收集数据: 所提供的文本文件。

(2) 准备数据: 用Python解析上述文件, 得到数值型数据。

- (3) 分析数据：用Tkinter构建一个GUI来展示模型和数据。
- (4) 训练算法：训练一棵回归树和一棵模型树，并与数据集一起展示出来。
- (5) 测试算法：这里不需要测试过程。
- (6) 使用算法：GUI使得人们可以在预剪枝时测试不同参数的影响，还可以帮助我们选择模型的类型。

Python有很多GUI框架，其中一个易于使用的Tkinter，是随Python的标准编译版本发布的。

这里说一下当时安装时的坑，其实 **Tkinter是自带在安装包里的，不需要重新安装**。如果非要 `pip` 一下的话，就会报错。



```
Anaconda Prompt

(C:\Users\Administrat\Anaconda3) C:\Users\Administrat>activate tensorflow
(tensorflow) C:\Users\Administrat>pip install tkinter
Collecting tkinter
  ERROR: Could not find a version that satisfies the requirement tkinter (from versions: none)
ERROR: No matching distribution found for tkinter

(tensorflow) C:\Users\Administrat>
```

<https://blog.csdn.net/TeFuirnever>

9、总结

数据集中经常包含一些复杂的相互关系，使得输入数据和目标变量之间呈现非线性关系。对这些复杂的关系建模，一种可行的方式是使用树来对预测值分段，包括分段常数或分段直线。一般采用树结构来对这种数据建模。相应地，若叶节点使用的模型是分段常数则称为回归树，若叶节点使用的模型是线性回归方程则称为模型树。

CART算法可以用于构建二元树并处理离散型或连续型数据的切分。若使用不同的误差准则，就可以通过CART算法构建模型树和回归树。该算法构建出的树会倾向于对数据过拟合。一棵过拟合的树常常十分复杂，剪枝技术的出现就是为了解决这个问题。两种剪枝方法分别是预剪枝（在树的构建过程中就进行剪枝）和后剪枝（当树构建完毕再进行剪枝），预剪枝更有效但需要用户定义一些参数。

Tkinter是Python的一个GUI工具包。虽然并不是唯一的包，但它最常用。利用Tkinter，可以轻松绘制各种部件并灵活安排它们的位置。另外，可以为Tkinter构造一个特殊的部件来显示Matplotlib绘出的图。所以，Matplotlib和Tkinter的集成可以构建出更强大的GUI，用户可以以更自然的方式来探索机器学习算法的奥妙。

本章是回归的最后一章，接下来将离开监督学习的岛屿，驶向无监督学习的未知港湾。在回归和分类（监督学习）中，目标变量的值是已知的。在后面的章节将会看到，无监督学习中上述条件将不再成立。下一章的主要内容是K-均值聚类算