决策树算法与python——心脏病预测

链接：<https://blog.csdn.net/qq_46488554/article/details/120871053>

## 一、介绍

数据来源于kaggle的Heart Failure Prediction 的数据集。

心血管疾病是全球头号死因，估计每年有1790万人丧生，占全球死亡总数的31%。心力衰竭是 CVD 引起的常见事件，此数据集包含 11 个可用于预测可能的心脏病的功能。

心血管疾病患者或心血管风险高的人（由于存在一种或多个危险因素，如高血压、糖尿病、高脂血症或已经建立的疾病）需要早期发现和管理，其中机器学习模型可以有很大的帮助。

变量属性：

1、Age：患者年龄[年]

2、Sex：患者的性别 [M： 男性， F： 女性]

3、ChestPainType： 胸痛类型 [TA： 典型的心绞痛， ATA： 非典型心绞痛， NAP： 非神经疼痛， ASY： 无症状]

4、RestingBP：休息血压[mm Hg]

5、Cholesterol：血清胆固醇 [mg/dl]

6、FastingBS： 禁食血糖 [1： 如果禁食> 120 毫克 / 分升， 0： 否则]

7、RestingECG：静息心电图结果 [正常： 正常， ST： 有 ST-T 波异常 （T 波反转和/或 ST 升高或凹陷 > 0.05 mV），LVH： 显示可能或明确的左心室肥大根据Estes的标准]

8、MaxHR：实现的最大心率 [60 至 202 之间的数字值]

9、ExerciseAngina：运动引起的心绞痛 [Y： 是的， N： 否]

10、Oldpeak： 相对于休息来说运动引起的ST段抑制 解释:(http://heart.dxy.cn/article/143557) [在抑郁症中测量的数字值]

11、ST\_Slope：峰运动ST段的坡度[向上：向上倾斜，平：平，向下：向下倾斜]

13.HeartDisease： 输出类 [1： 心脏病， 0： 正常]

## 二、过程

### 1.引入库

使用的相关库介绍：

（1）机器学习库sklearn，可以简化建模流程。

（2）Pandas 可以对各种数据进行运算操作，比如归并、再成形、选择，还有数据清洗和数据加工特征。

（3）NumPy是 Python语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。

**from sklearn import tree #决策树模型**

**from sklearn.model\_selection import train\_test\_split #训练集与测试集划分**

**import pandas as pd**

**import numpy as np**

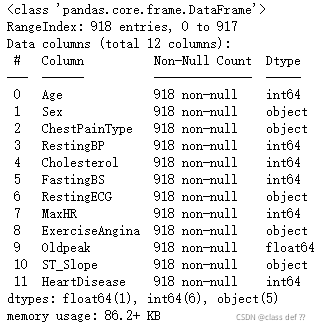
### 2.数据处理

读取数据并且述整体信息

data=pd.read\_csv('D:/心脏病预测/heart.csv')

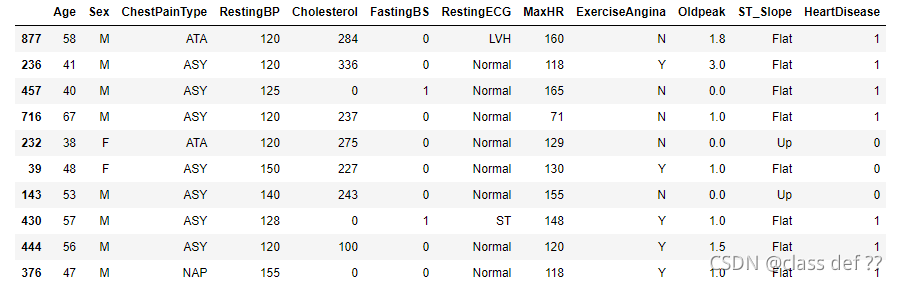
data.info()

得到下列信息：数据总共918行，共12列，总体无缺失值。数据类型：一个浮点，6个整数，5个字符串。



**data.sample(10)**

随机查看10条数据



可以看到数据基本干净而且清晰。

根据决策树的数据集要求，需要对以下变量做处理：

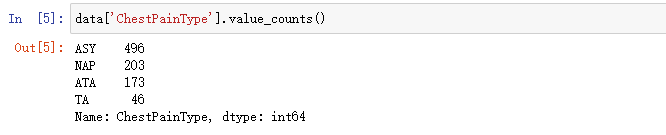
Sex,ChestPainType,RestingECG,ExerciseAngina,ST\_Slope，这些名义型变量都需要做数值映射。

Age，RestingBP，Cholesterol，MaxHR，Oldpeak，这些连续型变量都需要做数值替换。

### 3数据清洗与映射

### 关于名义数据映射操作：

1.先看“ChestPainType”取值，总共有哪些类型：



value\_counts()方法，统计变量值类型。

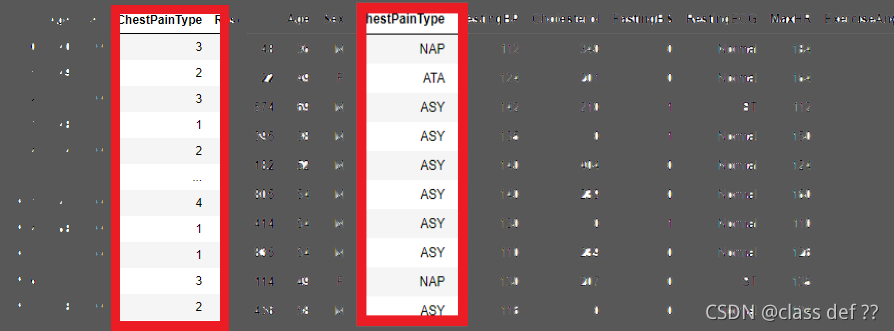
例如：ChestPainType中，共有4种变量值类型，其中“ASY”出现频次为496次。

2.再根据变量值的类型，建立映射字典。

就拿”Sex“这一变量为例，”M“男性就映射成0，”F“女性就映射成1.

3.最后使用map方法进行映射

'ChestPainType’列映射后于映射前的对比：



这五个名义值数据类型处理思路并没有较大的出入。

直接上代码，详细信息看注释：

**#对五个名义变量进行映射**

**data['ChestPainType'].value\_counts() #统计变量值类型**

**ChestPainType\_Map={'ASY':1,'NAP':2,'ATA':3,'TA':4} #建立 变量值类型：映射数值 的字典，就比如'ASY'这里映射成1.**

**data['ChestPainType']=data['ChestPainType'].map(ChestPainType\_Map) #进行变量值替换**

**data['Sex'].value\_counts()**

**Sex\_Map={'M':0,'F':1}**

**data['Sex']=data['Sex'].map(Sex\_Map)**

**data['RestingECG'].value\_counts()**

**RestingECG\_Map={'Normal':0,'LVH':1,'ST':2}**

**data['RestingECG']=data['RestingECG'].map(RestingECG\_Map)**

**data['ExerciseAngina'].value\_counts()**

**ExerciseAngina\_Map={'N':0,'Y':1}**

**data['ExerciseAngina']=data['ExerciseAngina'].map(ExerciseAngina\_Map)**

**data['ST\_Slope'].value\_counts()**

**Flat\_Map={'Flat':0,'Up':1,'Down':2}**

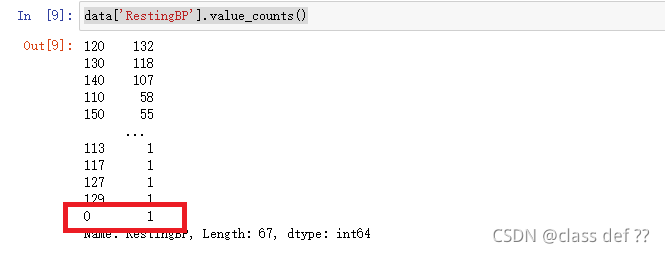
**data['ST\_Slope']=data['ST\_Slope'].map(Flat\_Map)**

关于连续型变量的映射操作：

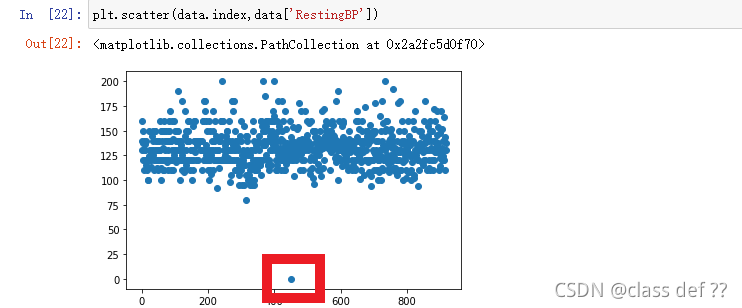
以’RestingBP’变量为例，

**data['RestingBP'].value\_counts()**

看下变量取值情况



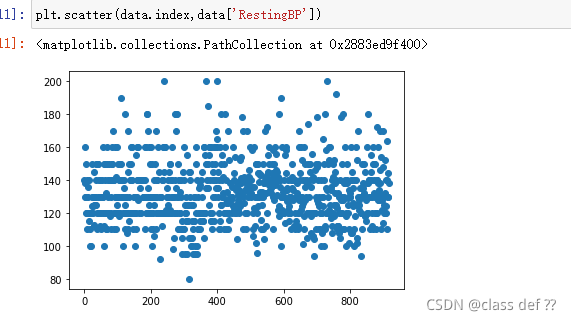
发现前面都很正常哈，就有一个非常离谱，属于异常数据。再用pyplot画个散点图，很容易就看到了：



因为只有一条，所以删除这条异常数据，不会对总体造成多大影响drop方法删除指定索引行。这里先提取’RestingBP’列’RestingBP’值为0的索引，找到后再删除。

**data=data.drop(data['RestingBP'][data['RestingBP']==0].index)**

再看眼数据，是正常的了。



在网站上了解搜索关于’RestingBP’[休息血压](https://www.youlai.cn/video/article/408041.html)后发现，该取值为90mm Hg到140mm Hg为正常。于是据此打算划分为三类，一类是偏高，二类是正常，三类是偏低。  
代码如下：

**def apply\_Resting(RestingBP):#小于90，返回0，大于140，返回2，中间为1**

**if RestingBP<90:**

**return 0**

**elif RestingBP>140:**

**return 2**

**else:return 1**

**data['RestingBP']=data['RestingBP'].apply(apply\_Resting)**

定义一个带参数的替换规则函数，然后apply  
apply是pandas非常灵活的一个方法，相当于遍历行或者列，对数据操作（参数设置）  
关于apply说明：

pandas 的 apply() 函数可以作用于 Series 或者整个 DataFrame，功能也是自动遍历整个 Series 或者DataFrame, 对每一个元素运行指定的函数。

接下来是’Cholesterol’血清胆固醇，百度词条。

虽然这个变量在百度词条里，根据年龄阶段有不同标准，

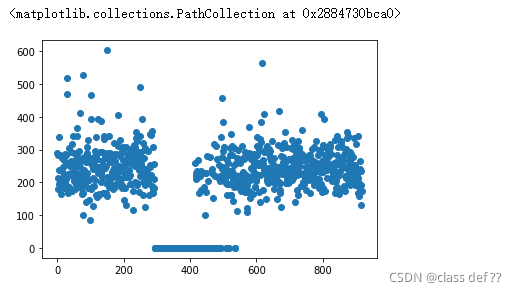
但我发现这些数据的年龄，都是分布在成年人范围

所以就按成年人标准划分，假如数据年龄分布包括小孩什么的，要严谨一点的话，还是要根据年龄再分不同标准。

在词条里，成年人的血清胆固醇为110-230mg/dl。

看分布：

**plt.scatter(data.index,data['Cholesterol'])**



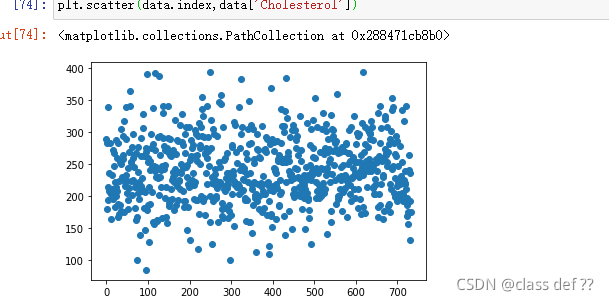
不知道这数据出了什么问题😓，三百多和不到五百多这里的大部分数据都变成了0。  
本人不是医学生，但感觉0肯定是数据错了，还有超过400以上也挺离谱  
好在异常数据较少，而且不会影响总体的分布，选择直接删除。

**data=data.drop(data['Cholesterol'][data['Cholesterol']==0].index)**

**data=data.drop(data['Cholesterol'][data['Cholesterol']>=400].index)**

**data.reset\_index(drop=True, inplace=True)**

再看，正常了



如法炮制，做映射

**def apply\_Cholesterol(Cholesterol):#小于110，返回0，大于230，返回2，中间为1**

**if Cholesterol<110:**

**return 0**

**elif Cholesterol>230:**

**return 2**

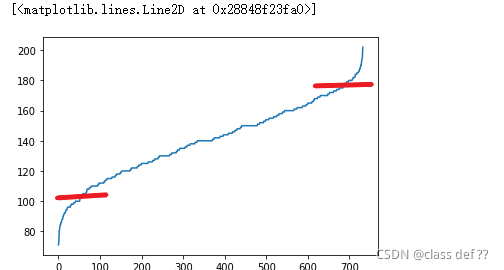
**else:return 1**

**data['Cholesterol']=data['Cholesterol'].apply(apply\_Cholesterol)**

"MaxHR"的处理

看了统计描述与分布图，数据没有问题。  
网上没有找到相关正常值的文章。  
于是决定根据统计规律划分，  
把这列大小排序后,画出折线图，发现有两个明显的转折点，就按这个分类。  
当然有更好的办法那就更好。

**plt.plot(data.index,data['MaxHR'][data['MaxHR'].sort\_values().index])**



**"Oldpeak"的处理**  
依旧是前面分析的思路，不赘述。  
"Oldpeak"分为四类。

**data=data.drop(data['Oldpeak'][data['Oldpeak']>4].index)**

**def apply\_Oldpeak(Oldpeak):**

**if Oldpeak<=1:**

**return 0**

**elif Oldpeak>1 and Oldpeak<=2:**

**return 2**

**elif Oldpeak>2 and Oldpeak<=3:**

**return 3**

**else: return 4**

**data['Oldpeak']=data['Oldpeak'].apply(apply\_Oldpeak)**

**“Age”的处理**  
数据年龄范围为28-77，根据国际标准划分，成熟期(29—40岁)、中年(41—65岁)、老年(66岁以后)。

**#划分年龄，成熟期40及以下为0，40到65之间为1，大于65为2**

**def apply\_Age(Age):**

**if Age<=40:**

**return 0**

**elif Age>40 and Age<=65:**

**return 1**

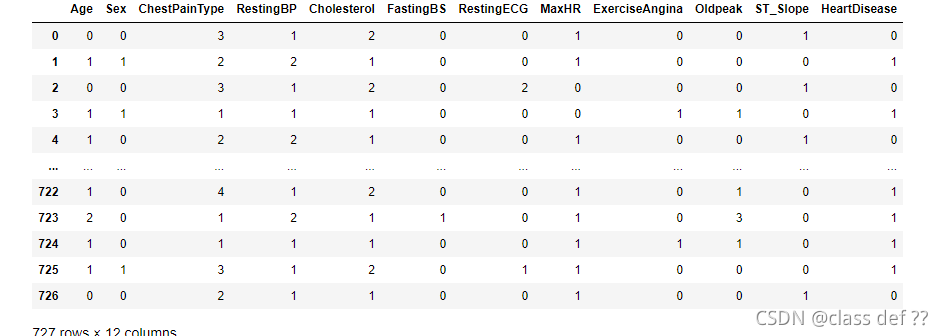
**else:return 2**

**data['Age']=data['Age'].apply(apply\_Age)**

重置一下序列

**data.reset\_index(drop=True, inplace=True)**

最终得到这份能直接建模的数据：



## 三.建模

### 1.决策树算法介绍

决策树(DecisionTree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。Entropy= 系统的凌乱程度，使用算法ID3, C4.5和C5.0生成树算法使用熵。这一度量是基于信息学理论中熵的概念。

简单来说决策树就是根据信息熵来划分类型，首先遍历所有按不同变量划分的结果，找到结果纯度最高的划法，再一步步循环这个过程，直到全部数据划分完。

纯度是个什么意思呢？举个例子[1,1,1,1,1]和[1,2,3,4,5]，第一个的纯度就是远远高于第二个的纯度。

这里纯度算法有常见的两种，信息熵和基尼指数，区别是信息熵越小越纯，基尼指数是越接近于0.5越好。

那怎么划分？以这个数据为例，以性别划分划分出来的信息熵，比按休息时血压划分的要大，理所当然选择第二种，于是决策树的第一个节点就算出来了。

### 2.拟合过程

拟合

因为分类目标变量包含再数据里面，于是提取出来，再在表里删除这个变量。

**target=data['HeartDisease']**

**data=data.drop('HeartDisease',1)**

把DataFrame表格转化成numpy数组，sklearn的数据类型需要。

**target=np.array(target)**

**data=np.array(data)**

训练集和测试集划分，  
train\_test\_split(数据,分类目标,测试集大小0-1)  
测试集过大过小都不利于建模准确和检验，一般建议0.3

**Xtrain, Xtest, Ytrain, Ytest=train\_test\_split(data,target,test\_size=0.3)**

建立模型，进行拟合，返回预测准确度

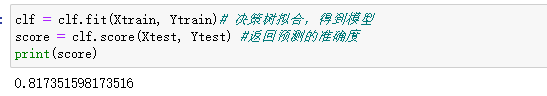
**clf = tree.DecisionTreeClassifier()# 载入决策树分类模型**

**clf = clf.fit(Xtrain, Ytrain)# 决策树拟合，得到模型**

**score = clf.score(Xtest, Ytest) #返回预测的准确度**

**print(score)**

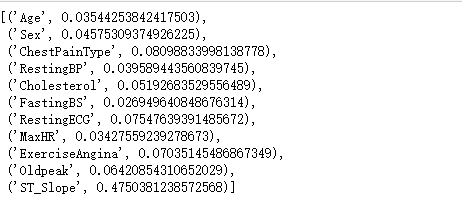
可以看到我这个拟合出来的某型准确度达到82%，算是比较好的成绩了



**clf.feature\_importances\_**

**[\*zip(feature\_name,clf.feature\_importances\_)]**

看看每个特征的重要程度



可以看到ST\_Slope这个特征非常重要，几乎占一半。

### 3可视化

**import graphviz**

导入graphviz包，还需要安装这个软件，windows注意在安装过程中要选择添加到系统变量中，然后重启电脑。

然后就是生成可视化图形啦。

clf：分类器

feature\_names:列名

class\_name:分类标签名

filled：是否填充颜色

rounded：图形边缘是否美化

**feature\_name=['Age','Sex','ChestPainType','RestingBP','Cholesterol','FastingBS','RestingECG','MaxHR','ExerciseAngina','Oldpeak','ST\_Slope']**

**dot\_data = tree.export\_graphviz(clf,**

**feature\_names= feature\_name,**

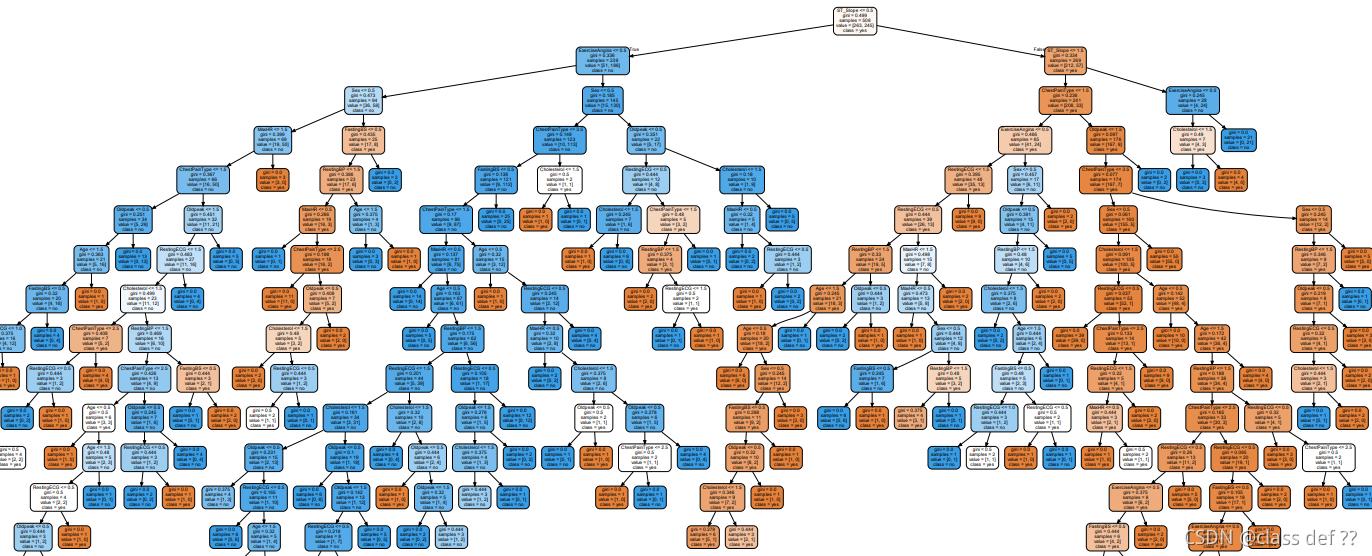
**class\_names=['yes','no'],**

**filled=True,rounded=True)**

**graph = graphviz.Source(dot\_data)#画树**

**graph.render('D:tree.pdf')**

得到图形



看起来是非常不方便的，而且你用训练集预测一下能有100%正确率，这其实是一种不好的现象，也就是过拟合，导致模型泛化能力较差，需要后续剪枝。

### 4.修正与优化

在你建立模型，也就是分类器的时候，是有大量参数是可以调的。  
例如：

clf = tree.DecisionTreeClassifier(criterion=‘gini’,  
max\_depth=4,  
max\_leaf\_nodes=10,  
min\_samples\_leaf=9,  
)

参数的设置可以提升模型的准确率与模型泛化能力。所以有很多段子说“调参侠”😂，调的就是这类东西。一个资深大师可以根据以往的经验，直接调到合适的参数。没达到一定境界就网格搜索，辅助调参。

网格搜索是一项模型超参数优化技术，常用于优化三个或者更少数量的超参数，本质是一种穷举法。对于每个超参数，使用者选择一个较小的有限集去探索。然后，这些超参数笛卡尔乘积得到若干组超参数。网格搜索使用每组超参数训练模型，挑选验证集误差最小的超参数作为最好的超参数。

%%time是jupyter notebook用来统计代码运行时长的

这里导入GridSearchCV

参数备选组成一个字典，比如 ‘criterion’:[‘gini’,‘entropy’],备选有“gini”和“entropy”两种。

GridSearchCV：

clf：模型

parameters：参数

refit：是否交叉验证训练集

cv:交叉验证参数

verbose：日志冗长度，int：冗长度，0：不输出训练过程，1：偶尔输出，>1：对每个子模型都输出。

n\_jobs：-1代表多核，建议启用，省时间

**%%time**

**from sklearn.model\_selection import GridSearchCV**

**Xtrain, Xtest, Ytrain, Ytest=train\_test\_split(data,target,test\_size=0.3)**

**clf = tree.DecisionTreeClassifier()# 载入决策树分类模型**

**parameters = {'max\_depth': [1,2,3,4,5,6,7,8,9],**

**'max\_leaf\_nodes':range(20),**

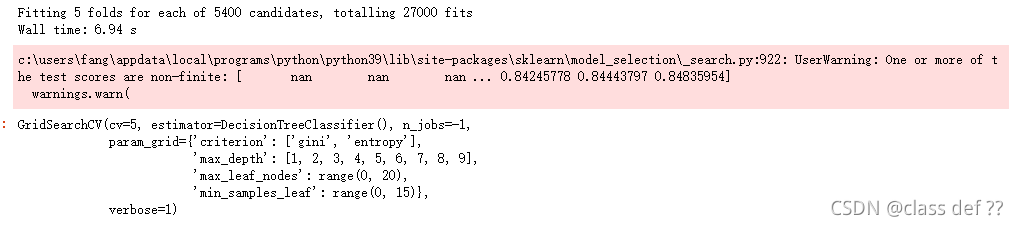
**'criterion':['gini','entropy'],**

**'min\_samples\_leaf':range(15)}**

**gs = GridSearchCV(clf, parameters, refit = True, cv = 5, verbose = 1, n\_jobs = -1)**

**gs.fit(Xtrain, Ytrain)**

运行得到，可以看到总共做了27000多次拟合，用时6.94s，单核的话估计好几分钟。



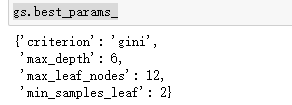
查看最优结果

**gs.best\_score\_**

请添加图片描述

查看最优参数

**gs.best\_params\_**



## 四、总结

带入上面的参数，

**Xtrain, Xtest, Ytrain, Ytest=train\_test\_split(data,target,test\_size=0.3)**

**clf = tree.DecisionTreeClassifier(criterion='gini',**

**max\_depth=6,**

**max\_leaf\_nodes=12,**

**min\_samples\_leaf=2,**

**)**

**clf = clf.fit(Xtrain, Ytrain)# 决策树拟合，得到模型**

**score = clf.score(Xtest, Ytest) #返回预测的准确度**

**import graphviz**

**feature\_name = ['Age','Sex','ChestPainType','RestingBP','Cholesterol','FastingBS','RestingECG','MaxHR','ExerciseAngina','Oldpeak','ST\_Slope']**

**dot\_data = tree.export\_graphviz(clf,**

**feature\_names= feature\_name,**

**class\_names=['yes','no'],**

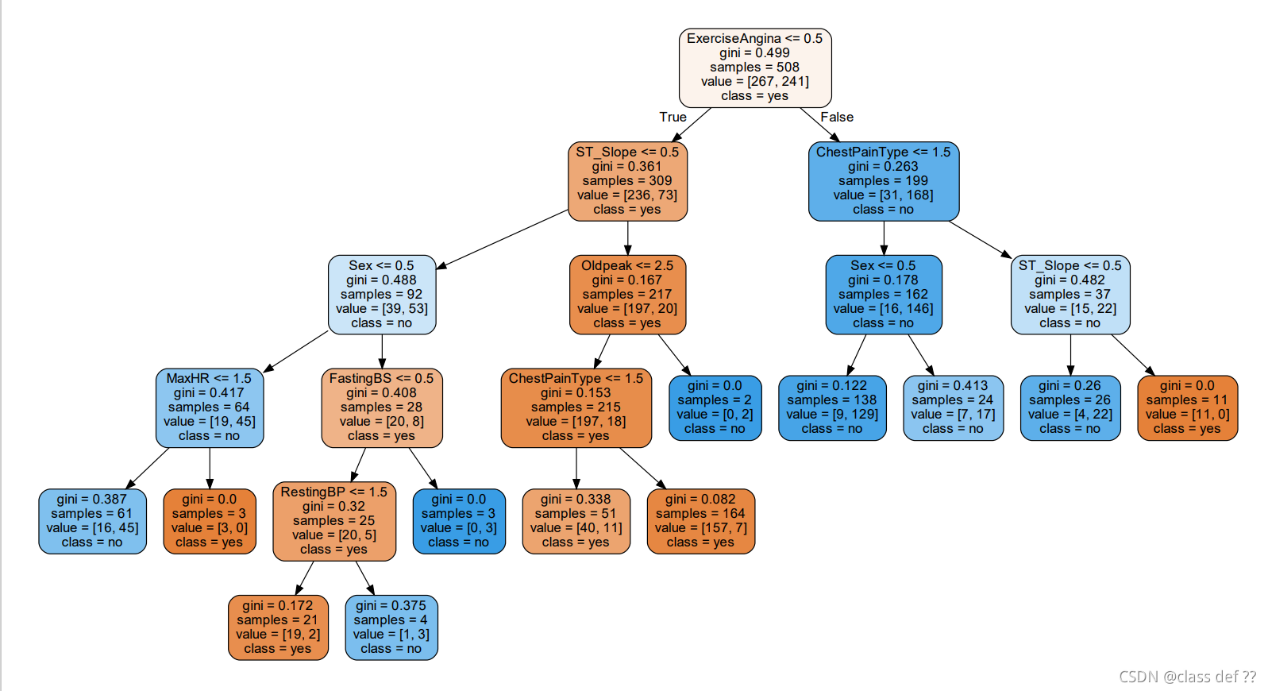
**filled=True,**

**rounded=True)**

**graph = graphviz.Source(dot\_data)#画树**

**graph.render('D:tree.pdf')**

最后生成决策树



看决策树，最高的是最重要的，节点包含划分条件，基尼指数，目前样本数，两类（分类标签多个的话就多个）样本数，最后分类结果。

比如看：

ExcerciseAngina为0，class=yes。

运动时引起的心绞痛为否，可能患心脏病。

很容易理解哈，不运动的时候都会心绞痛，那多半就是心脏病了。

再往下看。

ST\_Slope<=0.5,class=yes。

峰运动ST段的坡度为平的，再进一步增加患心脏病的可能性。

再往下看。

Sex<=0.5,class=no。

如果此时你的性别为女，那患心脏病的可能性再增加一步。

再往下看。

就是FastingBS。

最后可以简单的预测一下，指标全为0的人的心脏病预测结果为1，也就是患有心脏病。

