心脏病预测——决策树模型与 python

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 种可用于预测可能的心脏病的数据。

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。

11 种变量属性：

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 [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]

HeartDisease： 输出类 [1： 心脏病， 0： 正常]

二、过程

引入库介绍：机器学习库 sklearn，可以简化建模流程。

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。

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。

数据处理：

读取数据并且述整体信息

```
In [1]: from sklearn import tree
        from sklearn.model_selection import train_test_split
        import pandas as pd
        import numpy as np
        data=pd.read_csv('D:/心脏病预测/heart.csv')
        data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

随机查看 10 条数据

```
In [2]: data.sample(10)
```

Out[2]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
155	56	M	ASY	155	342	1	Normal	150	Y	3.0	Flat	1
13	49	M	ASY	140	234	0	Normal	140	Y	1.0	Flat	1
693	42	F	NAP	120	209	0	Normal	173	N	0.0	Flat	0
190	46	M	ASY	180	280	0	ST	120	N	0.0	Up	0
763	58	M	NAP	132	224	0	LVH	173	N	3.2	Up	1
617	57	M	ATA	124	261	0	Normal	141	N	0.3	Up	1
882	56	F	ATA	140	294	0	LVH	153	N	1.3	Flat	0
621	56	M	NAP	130	256	1	LVH	142	Y	0.6	Flat	1
827	43	F	NAP	122	213	0	Normal	165	N	0.2	Flat	0
301	55	M	ATA	140	0	0	ST	150	N	0.2	Up	0

可以观察到数据基本干净且清晰

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

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：

```
In [4]: data['ChestPainType'].value_counts()

Out[4]: ASY    496
        NAP    203
        ATA    173
        TA      46
        Name: ChestPainType, dtype: int64
```

ChestPainType 中，共有 4 种变量值类型，其中“ASY”出现频次为 496 次、“NAP”出现 203 次、“ATA”出现 173 次、 “TA” 出现 46 次。

2、再根据变量值的类型，建立映射字典。
以”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。
4，这五个名义值数据类型处理思路相似，可以一起处理。

```
In [5]: #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}
data['ChestPainType']=data['ChestPainType'].map(ChestPainType_Map)

data['Sex'].value_counts()
Sex_Map={'M':0,'F':1}
sssSZssdata['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：

1、以’RestingBP’变量为例：

```
In [2]: data['RestingBP'].value_counts()
```

```
Out[2]: 120    132
        130    118
        140    107
        110     58
        150     55
               ...
        113      1
        164      1
        117      1
        127      1
        0        1
        Name: RestingBP, Length: 67, dtype: int64
```

发现了一个异常数据。

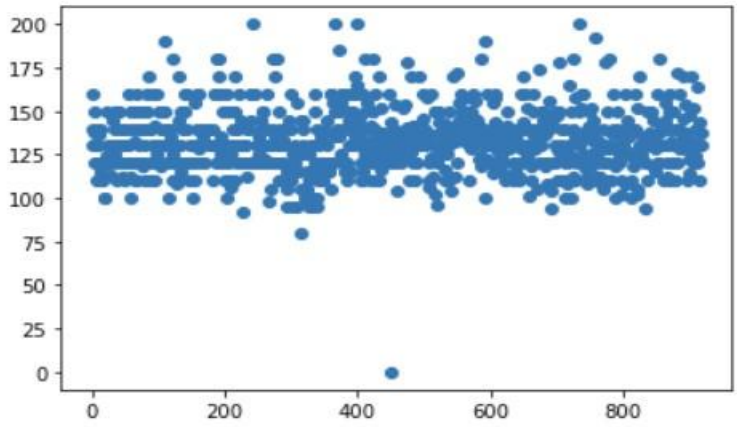
休息时心率不可能为 0。

再用 pyplot 画个散点图：

找到并且删除异常数据

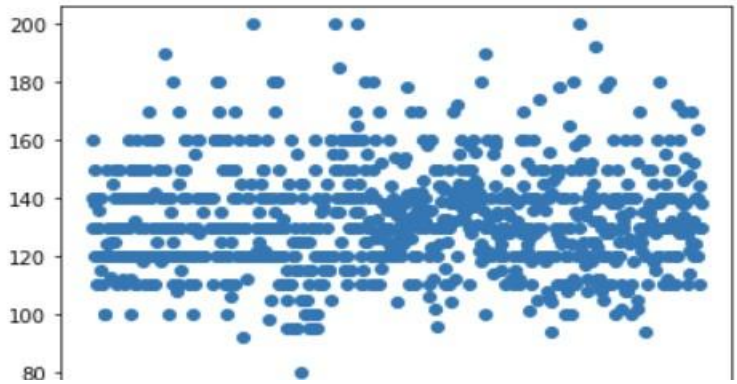
```
In [3]: plt.scatter(data.index,data['RestingBP'])
```

```
Out[3]: <matplotlib.collections.PathCollection at 0x2276affce50>
```




```
In [4]: data=data.drop(data['RestingBP'][data['RestingBP']==0].index)
        plt.scatter(data.index,data['RestingBP'])
```

```
Out[4]: <matplotlib.collections.PathCollection at 0x2276b3884f0>
```



根据资料显示，'RestingBP'休息血压，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

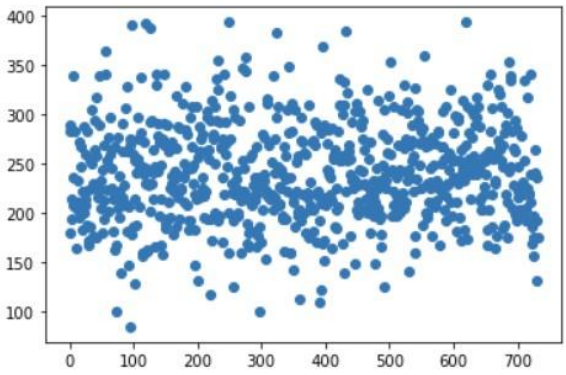
In [5]:
```
def apply_Resting(RestingBP):#小于90，返回0，大于140，返回2，中间为1
    if RestingBP<90:
        return 0
    elif RestingBP>140:
        return 2
    else:return 1

data['RestingBP']=data['RestingBP'].apply(apply_Resting)
```

’Cholesterol’血清胆固醇。
这个变量根据年龄阶段有不同标准，所以本次就按照成年人的标准划分，成年人的血清胆固醇为 110-230mg/dl。
对 Cholesterol 做映射并查看分布图

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

ut[26]: <matplotlib.collections.PathCollection at 0x1f346d0c8e0>



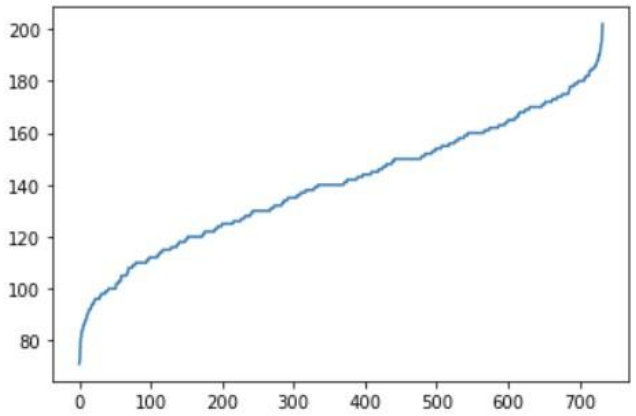
"MaxHR"的处理

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。

```
In [9]: plt.plot(data.index,data['MaxHR'][data['MaxHR'].sort_values().index])
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x2276b4a1df0>]
```



"Oldpeak"和“Age”的处理：

依旧是前面分析的思路。

"Oldpeak"分为四类。

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)

```
In [10]: 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)
```

```
In [11]: #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，最终得到：

```
data['Age']=data['Age'].apply(apply_Age)

data.reset_index(drop=True, inplace=True)

print(data)
```

```
tset2
D:\Python\python.exe D:/爬虫代码/心脏病预测/tset2.py
     Age  Sex  ChestPainType  ...  Oldpeak  ST_Slope  HeartDisease
0      0    0              3  ...        0         1             0
1      1    1              2  ...        0         0             1
2      0    0              3  ...        0         1             0
3      1    1              1  ...        2         0             1
4      1    0              2  ...        0         1             0
..   ...  ...            ...  ...      ...       ...           ...
722    1    0              4  ...        2         0             1
723    2    0              1  ...        4         0             1
724    1    0              1  ...        2         0             1
725    1    1              3  ...        0         0             1
726    0    0              2  ...        0         1             0

[727 rows x 12 columns]
```

三、建模

1、关于决策树算法的定义：

根据百度词条显示：

决策树算法是一种逼近离散函数值的方法。它是一种典型的分类方法，首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。本质上决策树是通过一系列规则对数据进行分类的过程。

决策树方法最早产生于上世纪60年代，到70年代末。由J Ross Quinlan提出了ID3算法，此算法的目的在于减少树的深度。但是忽略了叶子数目的研究。C4.5算法在ID3算法的基础上进行了改进，对于预测变量的缺值处理、剪枝技术、派生规则等方面作了较大改进，既适合于分类问题，又适合于回归问题。

决策树算法构造决策树来发现数据中蕴涵的分类规则．如何构造精度高、规模小的决策树是决策树算法的核心内容。决策树构造可以分两步进行。第一步，决策树的生成：由训练样本集生成决策树的过程。一般情况下，训练样本数据集是根据实际需要有历史的、有一定综合程度的，用于数据分析处理的数据集。第二步，决策树的剪枝：决策树的剪枝是对上一阶段生成的决策树进行检验、校正和修下的过程，主要是用新的样本数据集（称为测试数据集）中的数据校验决策树生成过程中产生的初步规则，将那些影响预衡准确性的分枝剪除。

2、拟合过程：

建立模型，进行拟合，返回预测准确度

In [2]:
```
target=data['HeartDisease']
data=data.drop('HeartDisease',1)

target=np.array(target)
data=np.array(data)
Xtrain, Xtest, Ytrain, Ytest=train_test_split(data,target,test_size=0.3)


clf = tree.DecisionTreeClassifier()
clf = clf.fit(Xtrain, Ytrain)
score = clf.score(Xtest, Ytest)
print(score)
```

```
0.7990867579908676
```

由此可见，拟合出来的某型准确度达到 80%

3、查看每个数据的重要程度

```
In [10]: feature_name = ['Age','Sex','ChestPainType','RestingBP','Cholesterol','FastingBS','RestingECG','MaxHR','ExerciseAngina','Oldpeak','ST_Slop
         clf.feature_importances_
         [*zip(feature_name,clf.feature_importances_)]

Out[10]: [('Age', 0.03177958068512582),
          ('Sex', 0.050604030596295646),
          ('ChestPainType', 0.11472210654936112),
          ('RestingBP', 0.040473686361729516),
          ('Cholesterol', 0.02535942536433879),
          ('FastingBS', 0.018777999735693166),
          ('RestingECG', 0.022425160464898966),
          ('MaxHR', 0.18951893650856308),
          ('ExerciseAngina', 0.08296851299930033),
          ('Oldpeak', 0.04943338042318447),
          ('ST_Slope', 0.373937180311509)]
```

可以看到最为重要的是“ST-Slope”最为重要

4、可视化：

导入 graphviz 包。

clf：分类器

feature_names:列名

class_name:分类标签名

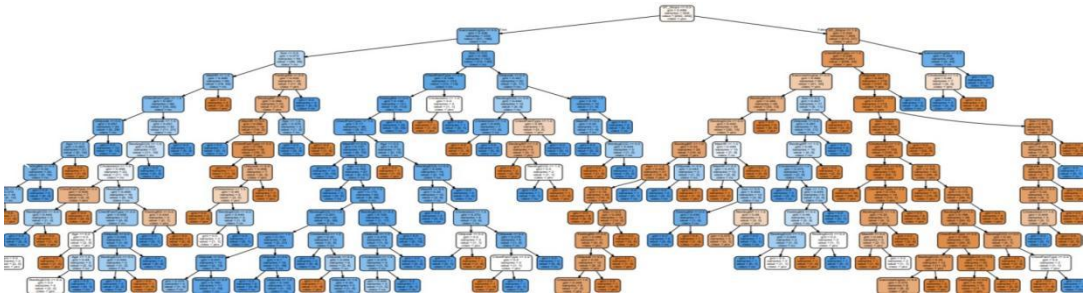
filled：是否填充颜色

rounded：图形边缘是否美化

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

得到图形



看起来是非常不方便的，而且用训练集预测一下能有 100%正确率，过拟合了，导致模型泛化能力较差，需要修正与优化。

5、本次使用网格搜素来进行辅助调参。

根据百度词条显示---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)
```

找到最优参数：

```
gs.best_params_
```

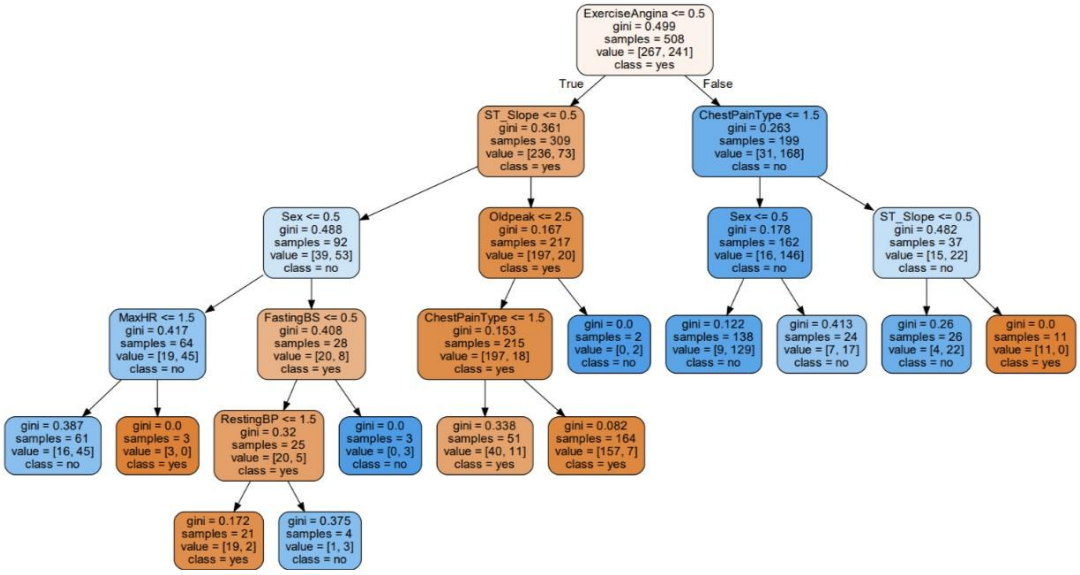
```
{'criterion': 'gini',
 'max_depth': 6,
 'max_leaf_nodes': 12,
 'min_samples_leaf': 2}
```

带入上面的参数

6、

```
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)
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。

例如：

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。

```
: clf.predict(np.zeros((1,11)))
: array([1], dtype=int64)
```