

数据挖掘设计报告

**ID3算法**

|  |  |  |  |
| --- | --- | --- | --- |
| **指标** | **评分细则** | **分值** | 得分 |
| 实现难度 | 1.算法步骤繁琐、理解困难。 | 30 |  |
| 界面 | 1.界面简洁，友好。 2.能选择输入数据 | 10 |  |
| 边界 | 1.算法边界清楚。 | 10 |  |
| 技术含量 | 1.软件的编写过程过中使用了较好的编程技巧。 | 10 |  |
| 论文质量 | 1.条理清楚、语言准确、图标清晰、格式正确 | 10 |  |
| 运行效果 | 1.算法基本能正常运行并且运行无错误。 | 30 |  |
| 总分 |  | 100 |  |

|  |  |
| --- | --- |
| 专业名称 | 计算机科学与技术 |
| 班级学号 | 210314 |
| 学生姓名 | 王贺麟 |
| 指导老师 | 鲁宁 |
| 提交时间 | 2024年5月24日 |

目录

[1 背景简介 5](#_Toc164679073)

[2 算法应用 6](#_Toc164679074)

[2.1 数据预处理 6](#_Toc164679075)

[2.2 算法流程图 6](#_Toc164679076)

[2.3 算法伪代码 6](#_Toc164679077)

[2.4 程序说明 6](#_Toc164679078)

[2.5 时间和空间复杂度 6](#_Toc164679079)

[2.6 算法边界 6](#_Toc164679080)

[3 运行结果 7](#_Toc164679081)

[4 结论 8](#_Toc164679082)

[5 附录 9](#_Toc164679083)

## 1 背景简介

ID3（Iterative Dichotomiser 3）算法是一种广泛使用的决策树算法，由Ross Quinlan在1986年提出。该算法的设计初衷是通过对数据进行分类以便更好地理解数据，并在机器学习和数据挖掘领域中得到了广泛应用。以下将从ID3算法的产生原因、基本原理以及涉及的名词定义等方面进行详细描述。

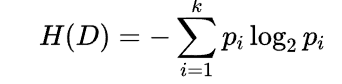
1.1 产生原因

在机器学习的初期，研究人员致力于开发能够自动从数据中提取知识的算法。决策树作为一种直观且易于解释的分类模型，迅速引起了研究人员的关注。然而，如何有效地构建决策树成为一个关键问题。ID3算法正是为了构建高效且易于理解的决策树而产生的。其核心思想是通过信息增益来选择最优特征，从而生成能够很好地对数据进行分类的决策树。

1.2 算法基本原理

ID3算法的核心思想是利用信息增益来选择划分数据集的特征。信息增益是从信息论中借鉴的概念，用来衡量某一特征对分类结果的不确定性的减少程度。具体步骤如下：

1. \*\*计算数据集的熵\*\*：熵（Entropy）是衡量数据集纯度的一种指标。对于一个分类任务，熵越低，数据集越纯。熵的计算公式为：



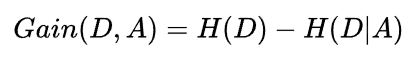
其中，D是数据集，k是类别数，p\_i是属于第i类的样本所占的比例。

1. \*\*计算特征的条件熵\*\*：条件熵是指在已知某一特征的条件下，数据集的熵。条件熵的计算公式为：



其中，A是某个特征，Values(A)是特征A的所有可能取值，D\_v是特征A取值为v的子数据集。

1. \*\*计算信息增益\*\*：信息增益是指由于特征A的引入而使得数据集D的不确定性减少的程度。信息增益的计算公式为：



4. \*\*选择最优特征\*\*：ID3算法选择信息增益最大的特征进行数据集的划分，并递归地在每个子数据集上重复上述步骤，直到所有特征都被使用完毕或子数据集中的样本全部属于同一类。

5. \*\*生成决策树\*\*：通过不断地选择最优特征进行划分，最终生成一棵决策树。决策树的每个节点代表一个特征，每个分支代表该特征的一个取值，叶子节点则代表最终的分类结果。

1.3名词定义

1. \*\*熵（Entropy）\*\*：用于衡量数据集纯度的指标，熵越低，数据集越纯。

2. \*\*信息增益（Information Gain）\*\*：特征引入后数据集不确定性的减少程度，信息增益越大，特征的区分能力越强。

3. \*\*条件熵（Conditional Entropy）\*\*：在已知某特征的条件下，数据集的熵。

4. \*\*决策树（Decision Tree）\*\*：一种树状结构，用于表示决策过程。内部节点表示特征，分支表示特征的取值，叶子节点表示分类结果。

5. \*\*递归（Recursion）\*\*：算法通过自身调用自身来解决问题的一种方法，ID3算法通过递归生成决策树。

1.4 应用与局限

ID3算法在分类任务中表现良好，特别是在处理离散数据时效果显著。然而，ID3算法也存在一些局限性，例如：

1. \*\*处理连续数据能力有限\*\*：ID3算法原生支持离散数据，处理连续数据需要预先进行离散化。

2. \*\*倾向多值特征\*\*：信息增益对多值特征有偏好，可能导致树的生成过于复杂。

总之，ID3算法作为一种经典的决策树算法，通过信息增益选择最优特征，逐步生成决策树，对数据进行有效分类。尽管存在一些局限性，但其直观、易于理解的特性使其在机器学习领域中具有重要地位。

## 2 算法应用

本章首先介绍算法具体应用过程，包括数据预处理、算法流程图、算法伪代码、程序说明，然后分析它的时间和空间复杂度，最后说明算法的边界。

### 数据预处理

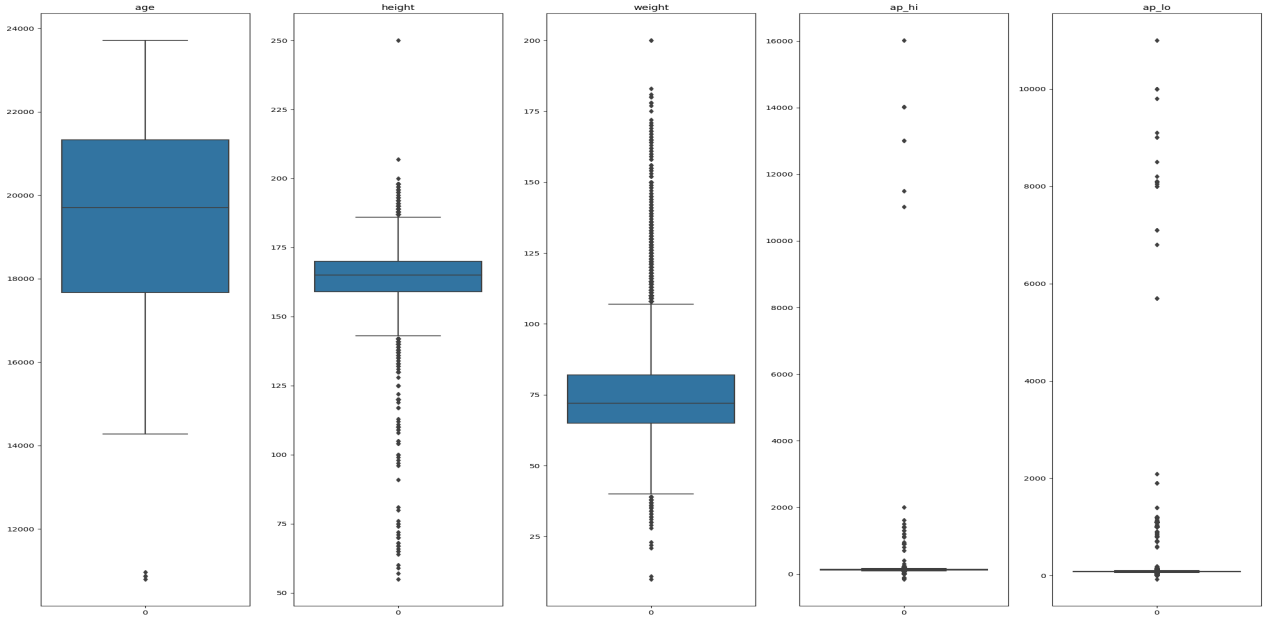
3.1.1数据描述性分析和正态性检验

首先，我们对样品的标签进行了分析，发现患病和未患病病例分别为49.97%和50.03%，表明完全均匀的样本量。其次，我们分析每个特征并绘制散点图在每个特征之间绘制图，以展示它们在样本中的关系。然后,计算各特征的统计指标，并进行正态性检。经检验，除ap hi和ap lo外，连续特征均遵循正态分布，样本分布均匀。

3.1.2缺失值和异常值处理

我们使用Python 的 pandas 库用于识别缺失值，由于缺失值很少我们直接删除所在行。

为了检测异常值，我们画了一个连续的箱形图。特征作为样本数据，直观地查看数据分布，



图：连续数据的箱线图

可以看出，异常值主要出现在身高、体重、舒张压和收缩压数据。 对分布有了一定的了解后对于异常值，我们结合IQR规则和基本常识来过滤筛选异常数据。 我们对异常值的定义标准如下：

• 身高和体重值超过上、下标准差 1.5 倍下四分位数被视为异常值。

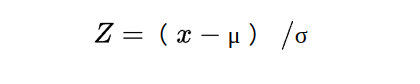
• 舒张压40mmHg，收缩压200mmHg以上，或收缩压低于80mmHg，且血压高于

250mmHg 被视为异常值。

根据上述标准，计算出异常样本数为约占样本总数的5.2%。 由于异常比例样本量不高，我们采用discard方法进行处理，直接删除总样本中的异常样本。

3.1.3 Z-SCORE归一化

仅对于SHAP分析，考虑到数据的尺度在不同的维度上是不同的，不能被分割直接比较，我们采用Z-SCORE归一化方法，以确保所有维度数据的大小是一样的。计算公式如下:

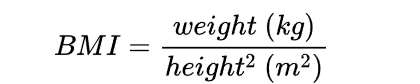


其中x为原始数据点，μ为数据集的平均值，σ为标准数据集的偏差。

3.1.4特征构建

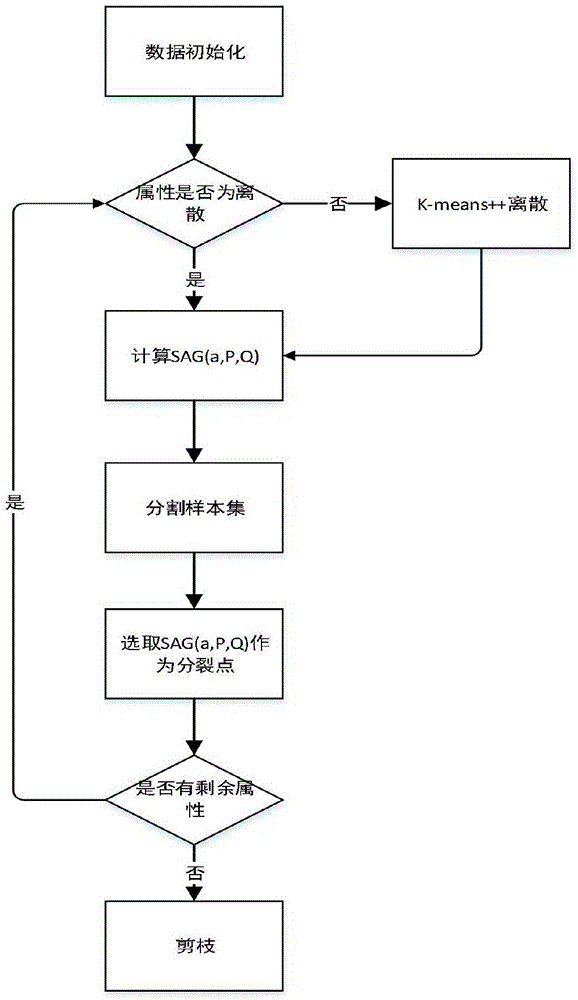
通过查阅相关文献，我们了解到体重是影响-的重要因素影响心血管疾病[1]。因此，根据身高和体重特征根据原始数据，我们引入了一个新功能，身体质量指数(BMI)，用来评估疾病的状况。计算公式如下:

体重指数

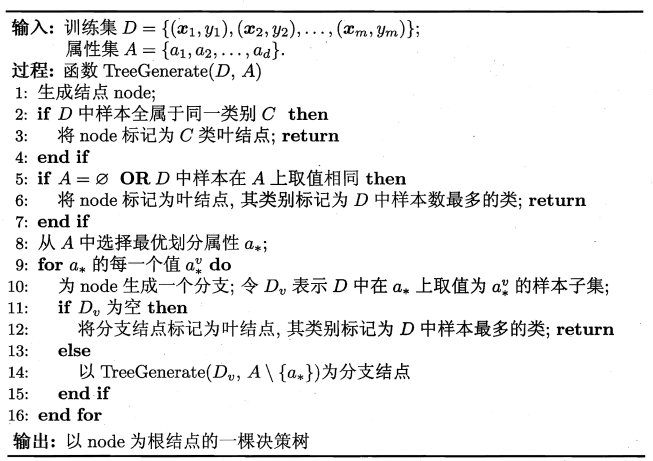


这里重量的单位是公斤，高度的单位是米

### 2.2 算法流程图



### 2.3 算法伪代码



### 2.4 程序说明

1.加载数据集并划分验证集和测试集合

from pycaret.datasets import get\_data

diabetes = get\_data('cardio\_train')

# init setup

from pycaret.classification import \*

clf1 = setup(data = diabetes, target = 'cardio')

2.定义树

# 构建ID3决策树

def id3(X, y, features, depth=0, max\_depth=None):

if len(np.unique(y)) == 1:

return y.iloc[0]

if len(features) == 0 or (max\_depth is not None and depth >= max\_depth):

return y.mode()[0]

best\_feature = best\_feature\_to\_split(X, y)

tree = {best\_feature: {}}

for value in np.unique(X[best\_feature]):

sub\_X = X[X[best\_feature] == value].drop(columns=[best\_feature])

sub\_y = y[X[best\_feature] == value]

subtree = id3(sub\_X, sub\_y, sub\_X.columns, depth + 1, max\_depth)

tree[best\_feature][value] = subtree

return tree

1. 十折交叉检验，训练树

# train logistic regression

dt = create\_model('dt',fold=10)

1. 验证性能

plot\_model(dtf, plot = 'confusion\_matrix')

plot\_model(dtf, plot = 'confusion\_matrix', plot\_kwargs = {'percent' : True})

5.遍历最大深度，进行剪枝

# 进行剪枝

def prune(tree, X, y):

def prune\_node(node, X, y):

if isinstance(node, dict):

feature = list(node.keys())[0]

for value in node[feature]:

if isinstance(node[feature][value], dict):

sub\_X = X[X[feature] == value].drop(columns=[feature])

sub\_y = y[X[feature] == value]

node[feature][value] = prune\_node(node[feature][value], sub\_X, sub\_y)

leaf\_counts = Counter(sub\_y)

most\_common\_label = leaf\_counts.most\_common(1)[0][0]

if all(isinstance(subtree, str) for subtree in node[feature].values()):

majority\_vote = Counter(node[feature].values()).most\_common(1)[0][0]

if majority\_vote == most\_common\_label:

return majority\_vote

return node

return prune\_node(tree, X, y)

# start a loop

models = []

results = []

for i in np.arange(5,50,1):

model = create\_model('dt', max\_depth = i)

model\_results = pull().loc[['Mean']]

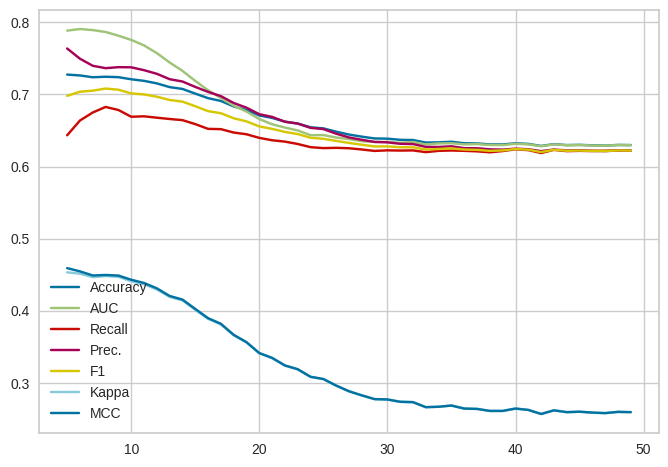
models.append(model)

results.append(model\_results)

results = pd.concat(results, axis=0)

results.index = np.arange(5,50,1)

results.plot()



得到最大深度为8左右，模型在测试集上效果最好

### 2.5 时间和空间复杂度

时间复杂度

ID3算法的时间复杂度主要受到以下因素的影响：

数据集中的实例数量（N）。

属性的数量（M）。

树的深度（D）。

主要时间开销在于计算信息增益和递归构建决策树。

计算信息增益：

需要计算每个属性的信息增益。

计算信息增益涉及计算熵，熵的计算是线性的，即O(N)。

对于M个属性，每个属性的计算复杂度为O(N)，总的计算复杂度为O(M \* N)。

递归构建决策树：

在最坏情况下，树的深度接近于N（每次只分一个实例）。

在每个层次上，计算信息增益的总复杂度为O(M \* N)。

因此，整个树构建过程的复杂度为O(M \* N \* D)，在最坏情况下，D ≈ N。

综上所述，ID3算法的最坏情况时间复杂度为O(M \* N^2)。

空间复杂度

ID3算法的空间复杂度主要受到以下因素的影响：

数据集的大小（N）。

属性的数量（M）。

树的深度（D）。

存储数据集：

数据集需要O(N \* M)的空间。

存储决策树：

决策树的深度在最坏情况下接近于N。

每个节点存储一个属性和若干子节点指针，总的空间复杂度接近于O(N)。

递归栈空间：

在最坏情况下，递归深度接近于N。

综上所述，ID3算法的空间复杂度为O(N \* M + N) ≈ O(N \* M)。

总结

时间复杂度： O(M \* N^2)（最坏情况）。

空间复杂度： O(N \* M)。

### 2.6 算法边界

适用于以下类型的数据：

分类问题： ID3算法主要用于分类问题。目标变量必须是离散的（即类标签），算法通过递归地选择最佳属性来构建决策树，从而对实例进行分类。

离散属性的数据： ID3算法最适合处理具有离散值的属性。每个属性可以有多个不同的离散值，这些值用于划分数据集。

属性间独立性： ID3假设属性之间是独立的。虽然在实际问题中这并不总是成立，但在属性相对独立时，ID3的效果较好。

不适用场景示例

回归问题： ID3不能处理目标变量为连续值的回归问题。

需要处理大量连续属性： 例如，股票价格预测中的时间序列数据。

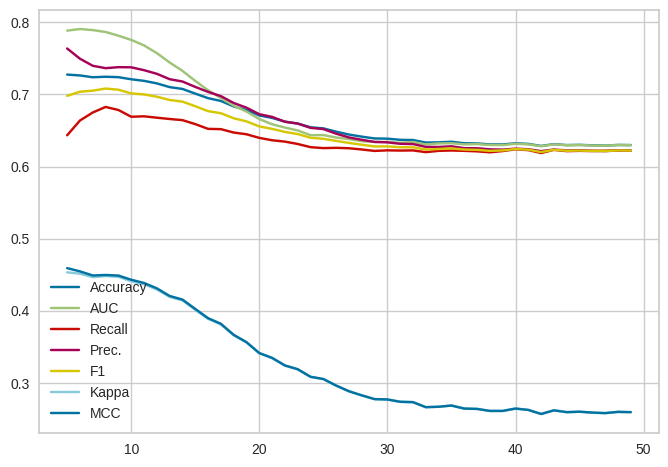
数据噪声较多的情况： 例如，传感器数据中存在大量干扰的情况下。

## 3 运行结果

1. 训练过程(10折交叉)

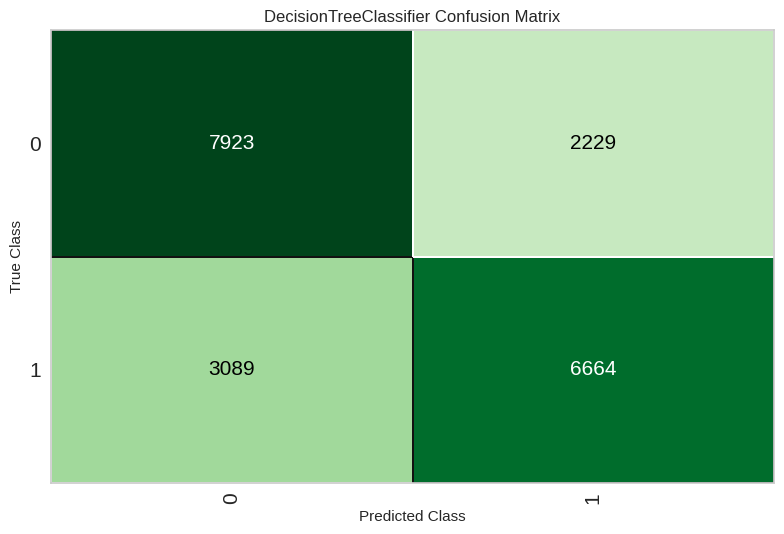
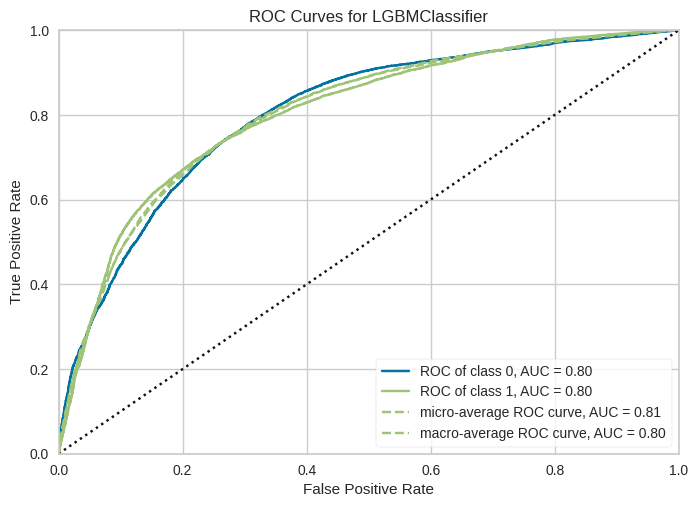


1. 剪枝



根据图中召回率和F1值的变化趋势选取最大深度为8

1. 模型效果

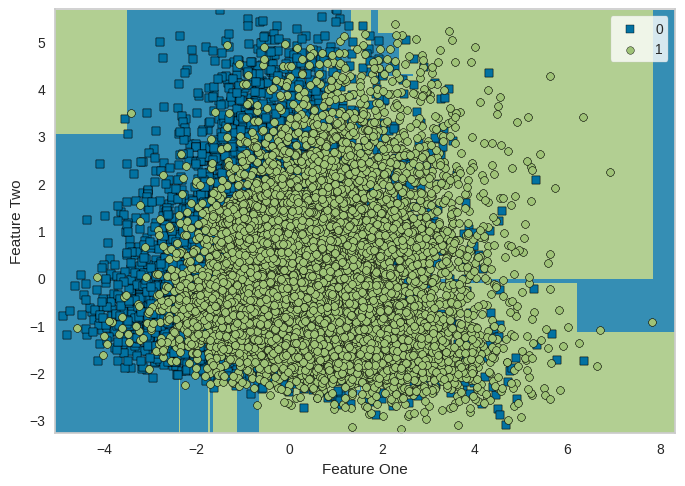
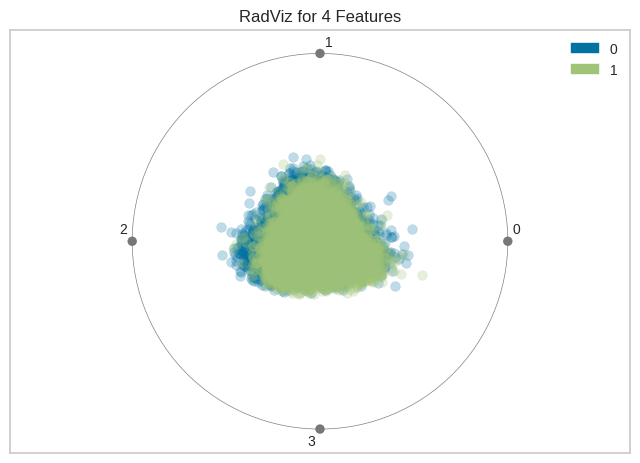
3.1混淆矩阵，ROC曲线和AUC值

3.2误差分析

错误样例中，1(有病)识别为0(未预测到疾病)的概率较高，整体的漏检率为16%

1. 模型解释

4.1决策边界



可见模型在2维(线性)决策边界不清晰，在边界内有诸多异常点，将维度使用扩展到4维可视化后也是如此，需要更高维度的特征参与(最终剪枝为8维，见后文树的可视化)

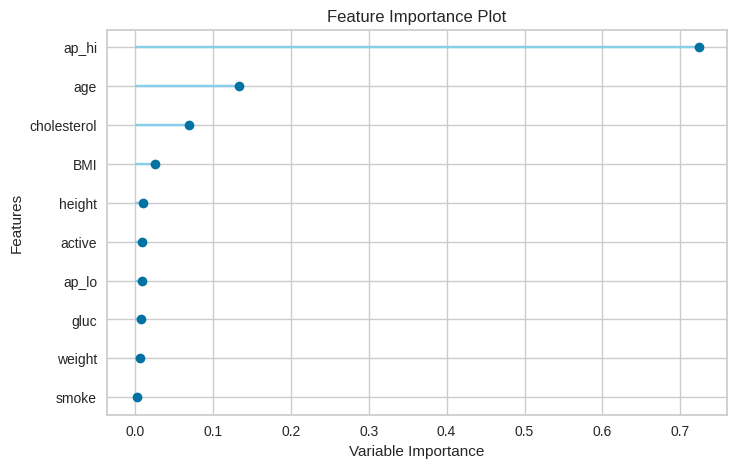
4.2树的可视化+解释特征重要性

4.2.1归一化后的树:



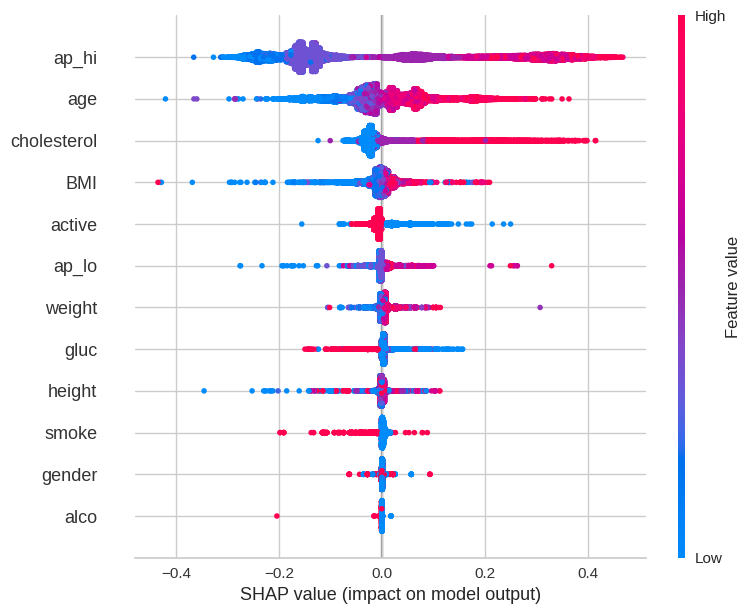
4.2.2 转化为原始数据的树:decision_tree_origin

4.2.3可视化特征的信息贡献率

其中重要特征有ap\_hi(极其重要),age,chlesterol

4.2.4使用SHAP补充解释特征重要性

SHapley加性解释(SHapley Additive explanation)是一种解释-的博弈论方法，使用任何机器学习模型(包括树模型、线性模型和神经网络模型)网络模型)。将最优信贷分配与局部解释联系起来博弈论中的物理SHAP值及其相关扩展



其中红点表示特征的较大值蓝点表示小数值。“+-”表示该特性的值与SHAP值分别呈正/负相关在SHAP图中红点和蓝点的贡献，可以推断，只有fol-以下特征对结果有一定影响。从张成的空间中推断在x轴上的分布，比较重要的特征是:

ap-hi(+), age(+), cholesterol(+), active(-) 与常识也匹配

## 4 结论

通过建模和分析，我们得出以下主要结论：

•模型的特征重要性ap\_hi>age>cholesterol>BMI>active,而其中除了active与是否患病是负相关外，其余都是正相关。即高收缩压、年龄、胆固醇和BMI偏大的人更易患病，而多运动的人更不易患病。其中收缩压跟是否患病的关系最大。

# 5 附录

附录A

核心代码

import pandas as pd

import numpy as np

from collections import Counter

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, roc\_curve, auc

import matplotlib.pyplot as plt

import seaborn as sns

# 加载数据

y = df.iloc[:, -2] # 倒数第二列为目标变量y

X = df.drop(columns=df.columns[-2]) # 其余列为特征变量x

# 计算熵

def entropy(y):

counts = Counter(y)

probabilities = [count / len(y) for count in counts.values()]

return -sum(p \* np.log2(p) for p in probabilities if p > 0)

# 计算信息增益

def information\_gain(X, y, feature):

original\_entropy = entropy(y)

values, counts = np.unique(X[feature], return\_counts=True)

weighted\_entropy = sum((counts[i] / sum(counts)) \* entropy(y[X[feature] == v]) for i, v in enumerate(values))

return original\_entropy - weighted\_entropy

# 选择最佳特征

def best\_feature\_to\_split(X, y):

features = X.columns

information\_gains = [information\_gain(X, y, feature) for feature in features]

return features[np.argmax(information\_gains)]

# 构建ID3决策树

def id3(X, y, features, depth=0, max\_depth=None):

if len(np.unique(y)) == 1:

return y.iloc[0]

if len(features) == 0 or (max\_depth is not None and depth >= max\_depth):

return y.mode()[0]

best\_feature = best\_feature\_to\_split(X, y)

tree = {best\_feature: {}}

for value in np.unique(X[best\_feature]):

sub\_X = X[X[best\_feature] == value].drop(columns=[best\_feature])

sub\_y = y[X[best\_feature] == value]

subtree = id3(sub\_X, sub\_y, sub\_X.columns, depth + 1, max\_depth)

tree[best\_feature][value] = subtree

return tree

# 进行剪枝

def prune(tree, X, y):

def prune\_node(node, X, y):

if isinstance(node, dict):

feature = list(node.keys())[0]

for value in node[feature]:

if isinstance(node[feature][value], dict):

sub\_X = X[X[feature] == value].drop(columns=[feature])

sub\_y = y[X[feature] == value]

node[feature][value] = prune\_node(node[feature][value], sub\_X, sub\_y)

leaf\_counts = Counter(sub\_y)

most\_common\_label = leaf\_counts.most\_common(1)[0][0]

if all(isinstance(subtree, str) for subtree in node[feature].values()):

majority\_vote = Counter(node[feature].values()).most\_common(1)[0][0]

if majority\_vote == most\_common\_label:

return majority\_vote

return node

return prune\_node(tree, X, y)

# 定义预测函数

def predict(tree, sample):

if not isinstance(tree, dict):

return tree

feature = list(tree.keys())[0]

if sample[feature] in tree[feature]:

return predict(tree[feature][sample[feature]], sample)

else:

return np.nan # 若特征值不在训练集中出现过，返回空

# 预测测试集

y\_pred = X\_test.apply(lambda x: predict(pruned\_tree, x), axis=1)

# 绘制混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

# 计算并绘制ROC曲线和AUC值

y\_prob = X\_test.apply(lambda x: predict(pruned\_tree, x), axis=1)

y\_test\_binary = y\_test.apply(lambda x: 1 if x == y.unique()[1] else 0) # 将类别转为0和1

y\_prob\_binary = y\_prob.apply(lambda x: 1 if x == y.unique()[1] else 0)

fpr, tpr, \_ = roc\_curve(y\_test\_binary, y\_prob\_binary)

roc\_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 7))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

# start a loop

models = []

results = []

for i in np.arange(5,50,1):

model = create\_model('dt', max\_depth = i)

model\_results = pull().loc[['Mean']]

models.append(model)

results.append(model\_results)

results = pd.concat(results, axis=0)

results.index = np.arange(5,50,1)

results.plot()