

**机器学习报告**

**题目： 决策树**

二级学院 计算机科学与工程学院

专 业 计算机科学与技术

班 级

学生姓名 学号

指导教师

时 间 2021.1.30

目 录

[目 录 I](#_Toc63682311)

[摘 要 II](#_Toc63682312)

[1、决策树算法基本原理 3](#_Toc63682313)

[1.1 决策树的基本概念 3](#_Toc63682314)

[1.2 决策树的生成 3](#_Toc63682315)

[1.2.1 信息熵和条件熵 3](#_Toc63682316)

[1.2.2 基尼指数 3](#_Toc63682317)

[1.2.3 ID3算法 4](#_Toc63682318)

[1.2.4 C4.5算法 4](#_Toc63682319)

[1.2.5 CART 算法 4](#_Toc63682320)

[1.2.5 剪枝 5](#_Toc63682321)

[2、决策树算法和实现 6](#_Toc63682322)

[2.1 算法的设计步骤 6](#_Toc63682323)

[2.2 算法的基础模块 9](#_Toc63682324)

[2.2.1 计算熵 9](#_Toc63682325)

[2.2.2 数据划分 9](#_Toc63682326)

[2.2.3 构建树模型 9](#_Toc63682327)

[2.2.4 剪枝 10](#_Toc63682328)

[2.2.5 使用决策树进行分类 10](#_Toc63682329)

[2.2.5 使用决策树进行预测 10](#_Toc63682330)

[2.2.5 使用决策树进行回归 10](#_Toc63682331)

[2.3 算法与Sklearn调包实现对比 10](#_Toc63682332)

[2.3.1 信息增益（使用鸢尾花数据集） 10](#_Toc63682333)

[2.3.2 基尼指数（使用鸢尾花数据集） 11](#_Toc63682334)

[2.3.3 回归树（使用波斯顿房价数据集） 12](#_Toc63682335)

[3、实战——泰坦尼克号 13](#_Toc63682336)

[3.1 数据预处理 13](#_Toc63682337)

[3.2 数据建模 16](#_Toc63682338)

[4、总结 16](#_Toc63682339)

[4.1 遇到的问题 16](#_Toc63682340)

[4.2 心得体会 17](#_Toc63682341)

[5、附录（代码） 18](#_Toc63682342)

摘 要

对决策树的算法进行了叙写，采用ID3/C4.5/CART算法进行决策树的构造。决策树分类方法采用自顶向下的递归方式，在每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。决策树回归方法将输入空间划分为M个区域，生成决策树。

关键词（3～5个）：决策树（Decision Tree）、ID3、C4.5、CART

# 1、决策树算法基本原理

# 1.1 决策树的基本概念

本文决策树是一种基本的分类和回归方法，是一种有监督学习，在机器学习中，决策树是一个预测模型。决策树是一种树形结构，其中每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。决策树学习包含三个过程：特征选择、决策树的生成和剪枝。决策树的生成是在当前节点上根据一定的评价标准选择数据中的某一属性作为该节点的分割，然后不断的进行递归操作，知道数据集中的所有特征都被判断完或者在节点上所有的数据都属于同一类，该过程结束。剪枝就是减去决策树中一些没有必要的枝，为了防止“过拟合”现象的发生。

## 1.2 决策树的生成

#### **1.2.1 信息熵和条件熵**

熵是表示随机变量不确定性的度量。选择某一个特征，也就是判断该特征是否能使信息的不确定性减少。

设X是一个取有限个值的离散随机变量，其概率分布P(X=xi)=pi,i=1,2,…n，则随机变量X的熵的定义为H(X)=-,(通常，该式的对数是以2为底或以e为底)，熵越大，随机变量的不确定性就越大。

设有随机变量(X,Y),其联合概率分布P(X=xi,Y=yi)=pij ,i=1,2,…n;j=1,2,…,m，条件熵H(Y|X)表示在已知随机变量X的条件下随机变量Y的不确定性，H(Y|X)=-=P(X=xi),i=1,2…n

当熵和条件熵中的概率由数据估计(特别是极大似然估计)得到时，所对应的熵和条件熵分别称为经验熵和经验条件熵。

信息增益的为表示得知特征X的信息使得类Y的信息的不确定性减少的程度。

特征A对训练数据集D的信息增益g(D,A),定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即：g(D,A)= H(D)- H(D|A) 。

#### **1.2.2 基尼指数**

假设有K个类，样本点属于第k类的概率为pk  ,则概率分布的基尼系数定义为：

Gini(p)=

样本集合D的基尼指数为Gini(D)=

(这里是D中属于第k类的样本子集，K是类的个数)

如果样本集合D根据特征A被分为D1和D2两部分，则特征A条件下集合D的基尼指数为： Gini(D,A)=

基尼指数和熵都可以表示数据的不确定性，不同的是基尼指数更偏向于连续属性，熵更偏向于离散属性

#### **1.2.3 ID3算法**

ID3算法流程：

1. 计算数据集的经验熵

H（D）=

1. 遍历所有的特征，按照信息增益最大的属性进行分裂

H(D|A)=

g(D,A)=H(D)-H(D|A)

1. 根据分裂属性划分样本
2. 重复上述流程，直至满足条件结束

ID3算法缺陷：

只能处理离散值

以信息增益作为划分训练数据集的特征，存在偏向于选择取值较多的特征的问题。

#### 1.2.4 C4.5算法

C4.5采用信息增益率来进行特征划分，可以理解为：信息增益率 = 惩罚参数 \* 信息增益

信息增益率为gR（D,A）=,其中HA(D)=

C4.5算法在ID3算法的基础上增加了处理连续值属性以及处理属性值缺失值的情况

连续值处理情况：

1. 连续值属性从小到大排序，每对相邻点的中点作为分裂点
2. 数据集D中有N个不同的连续值属性值，产生N-1个分裂点
3. 按照每个分裂点，计算每个二叉树的信息增益
4. 选择信息增益最大的

#### **1.2.5 CART 算法**

CART算法由两部分组成: （CART假设决策树是二叉树）

决策树的生成：生成决策树

决策树的剪枝：对生成的决策树进行剪枝并选择最优子树，用损失最下作为剪枝的标准

回归树的生成：

1. 对回归树采用平方误差最小化准则，选择平方误差最小的最优切分变量和切分点。
2. 用选定的切分点和切分特征划分区域，并决定相应的输出。
3. 继续对两个子区域调用步骤（1）和（2），直至满足停止条件。
4. 将输入空间划分为M个区域R1,R2…RM,生成决策树。

分类树：

1. 计算样本的基尼指数。
2. 遍历所有特征以及特征中的每一个取值，找出基尼指数最小的特征作为最优切分特征以及特征取值。

#### **1.2.5 剪枝**

剪枝分为预剪枝和后剪枝

剪枝是为了防止产生的决策树“过拟合”。

预剪枝是在决策树生成过程中，对每个结点在划分前先进行估计，若当前结点的划分不能带来决策树泛化性能的提升，则停止划分并将当前结点标记为叶节点

后剪枝则是先从训练集生成一颗完整的决策树，然后自底向上的对非叶结点进行考察，若将该结点对应的子树替换为叶节点能带来决策树泛化性能的提升，则将该子树替换为叶节点

预剪枝：

1. 当树达到一定深度的时候，停止树的生长
2. 当到达当前节点的样本数量小于某个阈值的时候，停止树的生长。
3. 计算每次分裂对测试集的准确度的提升，当小于某个阈值的时候，不再继续生长。

后剪枝：

1. REP（错误率降低剪枝）自底向上的对非叶结点进行考察，要判断若将该结点对应的子树替换为叶节点能否带来决策树泛化性能的提升。
2. CCP（代价复杂度剪枝）对于决策树T0中每一个内部结点，都可以计算g(t)=在树T0中减去g(t)最小的子树T(t)，(因为当α偏小，最优子树T α偏大，子树偏大意味着拟合程度更好。

# 2、决策树算法和实现

## 2.1 算法的设计步骤



图2.1.1 决策树分类树流程图

决策树的生成是一个递归过程，有两种情形会导致递归返回：

1. 当前结点包含的样本全属于同一类别，无需划分。
2. 当前结点包含的样本集为空，不能划分，同样把当前结点标记为叶结点。

ID3主要步骤：

1. 定义信息熵计算函数，计算数据集的信息熵。
2. 定义数据划分函数，用于根据指定特征的指定取值，划分数据集。

对于离散特征计算时，一个特征用于计算其对数据集的经验条件熵时，不需要此特征在子数据集中。

1. 遍历特征A的所有取值，计算数据集的经验熵H(D)以及特征A对数据集的经验条件熵H(D|A)。
2. 计算特征A对数据集的信息增益g(D,A)=H(D)-H(D|A)。
3. 以此类推,计算各特征对数据集的信息增益，取信息增益最大的特征Ag为最佳划分特征，对该特征的每一个可能的取值ai，依据Ag=ai将数据集D分割若干子集，得到树T。
4. 对树T各结点以各个结点的子集为训练集，以A-{Ag}为特征集，递归的调用step3-5,得到子树Ti，返回Ti。

C4.5主要步骤：

1. 定义信息熵计算函数，计算数据集的信息熵
2. 定义数据划分函数，用于根据指定特征的指定取值，划分数据集

对于离散特征计算时，一个特征用于计算其对数据集的经验条件熵时，不需要此特征在子数据集中。

对于连续值处理时，不需要删除该特征。此特征的其他取值在之后的构建树中还会用到

1. 遍历特征A的所有取值，计算数据集的经验熵H(D)以及特征A对数据集的经验条件熵H(D|A)

对于连续特征，采用二分法的数据处理方式，计算特征A下对应划分点的经验条件熵，找出最大的熵值所对应的划分点

1. 计算特征A对数据集的信息增益比为其信息增益熵与训练数据集D关于特征A的值的熵H(D)之比，即gR (D,A)=g(D,A)=H(D)-H(D|A)
2. 以此类推,计算各特征对数据集的信息增益之比，取信息增益之比最大的特征Ag为最佳划分特征，对该特征的每一个可能的取值ai，依据Ag=ai将数据集D分割若干子集，得到树T
3. 对树T各结点以各个结点的子集为训练集，以A-{Ag}为特征集，递归的调用step3-5,得到子树Ti，返回Ti

CART主要步骤：

1. 定义基尼指数计算函数，计算数据集的基尼指数
2. 定义数据划分函数，用于根据指定特征的指定取值，划分数据集

对于离散特征计算时，一个特征用于计算其对数据集的经验条件熵时，不需要此特征在子数据集中。

对于连续值处理时，不需要删除该特征。此特征的其他取值在之后的构建树中还会用到

1. 对于离散特征，遍历特征下的所有取值，数据集D被特征的某一取值分为两部分，计算出此特征条件下集合D的基尼指数

对于连续值，采用二分法计算基尼指数

1. 以此类推,计算各特征对数据集的基尼指数，取基尼指数最小的特征Ag为最佳划分特征，对该特征的每一个可能的取值ai，依据Ag=ai将数据集D分割为两个子集，得到树T
2. 对树T两个结点以各个结点的子集为训练集，以A-{Ag}为特征集，递归的调用step3-5,得到子树Ti，返回Ti

CART回归主要步骤：



图2.1.2 决策树回归树流程图

1. 定义空间划分函数，对应特征下的特征值小于特征划分点后为空间R1，大于则为空间R2
2. 定义mse计算函数，计算对应特征切分点的平方误差
3. 变量特征j下的所有特征取值s，求解
4. 遍历所有特征，求解最优切分点和切分遍历
5. 用选定的对（j，s）划分区域并决定相应的输出值
6. 继续将划分的两个子区域求最佳切分点，递归的调用step3-5，将输入空间划分为M个区域

## 2.2 算法的基础模块

#### **2.2.1 计算熵**

（1）循环每一个特征

（2）计算单个特征下每个特征值的数量

（3）选择算法（ID3/C4.5/CART），根据输入的特征确定是离散特征还是离散特征

ID3为离散值：

循环特征下的所有特征值，经验条件熵累加

计算信息增益，数据集的经验熵减去经验条件熵得出信息增益

循环所有特征，找出信息增益最大的，作为最有特征

C4.5为离散值：

与ID3相同，不同的是C4.5在计算完信息增益后需要计算信息增益率

C4.5为连续值：

对于特征值只有一个的特征，要跳出当前特征的循环，避免在计算信息增益率的时候，分母为0，找出连续特征中信息增益最大的划分点

注：决策树处理连续值

决策树处理的时离散数据，当离散数据中混杂着连续数据时，要将连续数据离散化。将连续取值的值域划分为多个区间，每个区间视为一个属性取值，这样就将连续数据离散化了。一般采用二分法，将特征的所有数据取值从小到大排列，我们可以取任意相邻取值的中位点，作为划分点，选择所有划分点信息增益最大的划分点。与离散属性不同，若当前结点划分属性为连续属性，该属性还可作为其后代结点的划分属性。

CART：

CART算法中选择的时基尼指数最小的

#### **2.2.2 数据划分**

对于离散特征，数据集划分的时候，要删除该特征，返回删除特征之后的数据集

对于连续特征，数据集划分的时候，分为小于划分点的数据和大于划分点的数据，不删除特征

#### **2.2.3 构建树模型**

如果数据集中所有实例属于同一类，则将类作为该结点类的标记

如果特征为空，则将数据集中实例数最大的类作为结点的类标记返回

根据计算熵的函数返回对应特征和最优划分点。

利用字典嵌套的方式存储树的信息，如果返回的特征是离散特征，则循环每一个属性值递归构建树。如果返回的特征是连续值，则树模型要分为左枝和右枝递归构建树模型。

每次构建之后，树模型的深度要加一。

#### **2.2.4 剪枝**

设定阈值：在构建树模型的时候，判断某一特征的划分能否为决策树带来性能的提升，如果不能，则停止划分，当前结点为叶节点，为数据集中类别最多的类。

设定树的参数：如果当前树的深度大于设定树的深度，则停止划分。

#### **2.2.5 使用决策树进行分类**

获取决策树的第一个结点，获取第一个结点所对应的键值，之后循环结点键值中所对应的值，如果为离散值，如果对应数据特征下的特征值为所对应的值，则判断该结点下特征值所对应的键值是否是字典，即判断是否有子节点，是的话进行递归，否的话得出结果。如果为连续值，需要提取对应的特征划分点，以及确定特征，类比于离散值，分为左枝和右枝，如果没有子节点，则返回结果。

#### **2.2.5 使用决策树进行预测**

根据测试集生成树模型，对测试集中的每一条数据进行循环，预测该树据的分类，将分类结果与实际结果进行比较，计算准确率。

#### **2.2.5 使用决策树进行回归**

循环所有特征，根据均方误差找出最优切割特征和最优切割点，根据选择的特征划分数据集，递归构建决策树。决策树中的回归树的回归值和分类树进行分类时类似，不同的是回归树在判断的时候衡量的是mse值.

## 2.3 算法与Sklearn调包实现对比

#### **2.3.1 信息增益（使用鸢尾花数据集）**

使用信息增益之后的决策树（使用相同的数据集）：

使用算法：

{'Petal.Length≤2.45': {'是': 0, '否': {'Petal.Length≤4.75': {'是': {'Petal.Width≤1.6': {'是': 1, '否': 2}}, '否': {'Petal.Length≤5.15': {'是': {'Petal.Width≤1.75': {'是': {'Sepal.Width≤2.35': {'是': 2, '否': {'Petal.Length≤5.05': {'是': 1, '否': {'Sepal.Length≤6.15': {'是': 1, '否': 2}}}}}}, '否': {'Sepal.Width≤3.1': {'是': 2, '否': 1}}}}, '否': 2}}}}}}

调包实现：

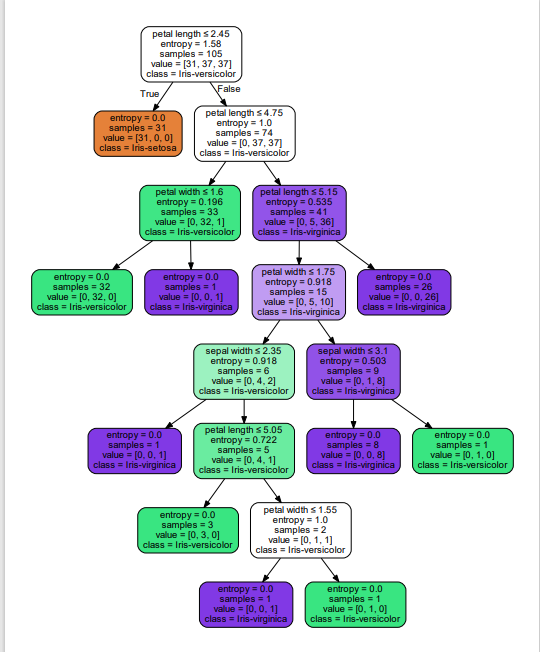
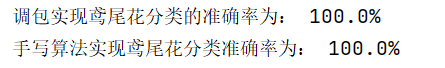


图2.3.1.1 sklearn调包实现鸢尾花分类（entropy）



根据对比，得出sklearn调包和算法实现的一样

#### **2.3.2 基尼指数（使用鸢尾花数据集）**

使用基尼指数之后的决策树（使用相同的数据集）：

算法：

{'Petal.Width≤0.8': {'是': 0, '否': {'Petal.Width≤1.75': {'是': {'Petal.Length≤4.95': {'是': {'Petal.Width≤1.6': {'是': 1, '否': 2}}, '否': {'Petal.Width≤1.55': {'是': 2, '否': {'Petal.Length≤5.449999999999999': {'是': 1, '否': 2}}}}}}, '否': {'Petal.Length≤4.85': {'是': {'Sepal.Width≤3.1': {'是': 2, '否': 1}}, '否': 2}}}}}}

调包实现：

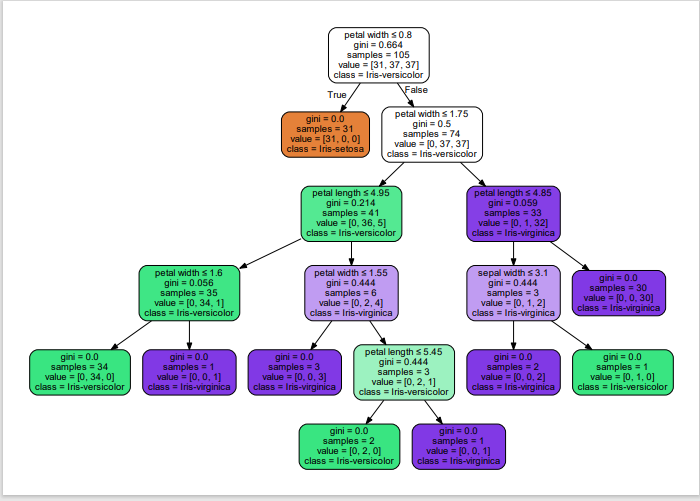
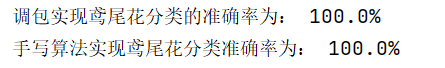


图2.3.1.2 sklearn调包实现鸢尾花分类（gini）



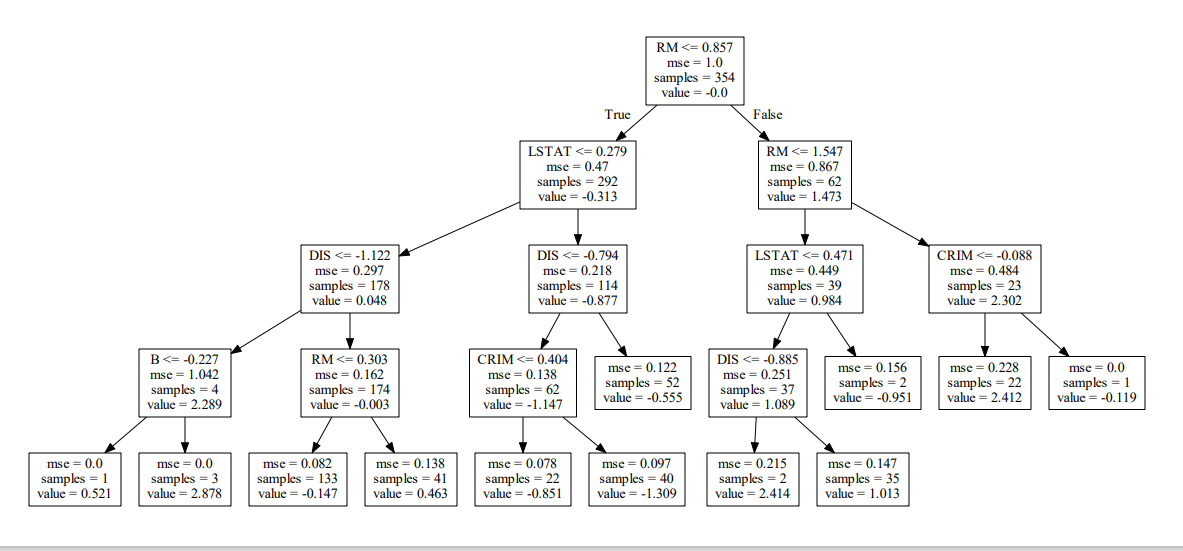
根据对比的，算法实现的决策树和使用sklearn实现效果一样

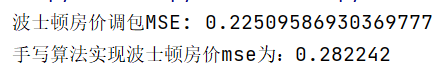
#### **2.3.3 回归树（使用波斯顿房价数据集）**

使用手写算法：

{'RM≤0.8539855444271394': {'是': {'LSTAT≤0.27439793377157': {'是': {'DIS≤-1.1347826417167073': {'是': {'CRIM≤0.577711781540726': {'是': 2.8782159413386963, '否': 0.5209617857893937}}, '否': {'RM≤0.29842570043375355': {'是': {'LSTAT≤-0.6605039465097802': {'是': {'CRIM≤-0.41425878536427857': {'是': 0.6062922529585992, '否': {'CHAS≤-0.28154625318573717': {'是': {'AGE≤-1.7266080322791177': {'是': {'ZN≤0.4693395644168946': {'是': {'INDUS≤-0.6094885648764784': {'是': {'CRIM≤-0.4113987444576826': {'是': 0.21163884230102378, '否': {'CRIM≤-0.39866283177214185': {'是': 0.17963991711257166, '否': 0.19030622550872248}}}}, '否': 0.11564206673566772}}, '否': 0.06231052475491408}}, '否': 0.010233842585472525}}, '否': 0.4249650102240376}}}}, '否': -0.19960660363945285}}, '否': 0.4632076281321877}}}}, '否': -0.8766053064325612}}, '否': 1.472843791529745}}

调包实现：





根据比对得：手写算法和调包实现，误差比较小.

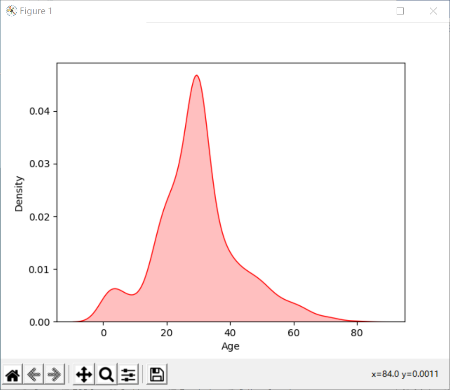
# 3、实战——泰坦尼克号

## 3.1 数据预处理

1.读取数据，查看数据的数据格式以及信息，以及查看数据的缺失情况

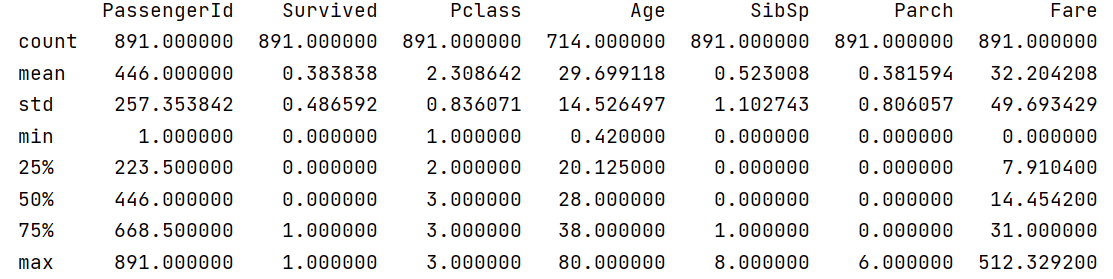
2.Cabin数据缺失过多，直接删除整列数据

绘制年龄的概率密度函数图



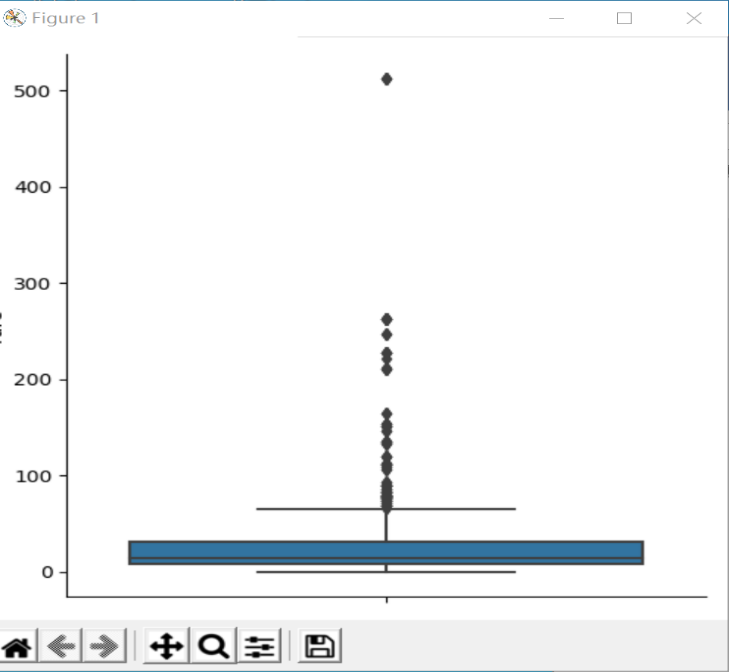
年龄的数据满足正态分布，将年龄的177条缺失值使用均值进行补充

将Embarked的缺失值使用数量最多值进行填充

根据数据的描述情况：

发现票价票价的最大值是512,而均值却是32，所以可能存在异常值

所以绘制票价的箱型图



票价存在着异常值，但是也不排除有类似于头等舱的情况

将数据中性别进行离散化处理，将男性转为1，女性转为0。

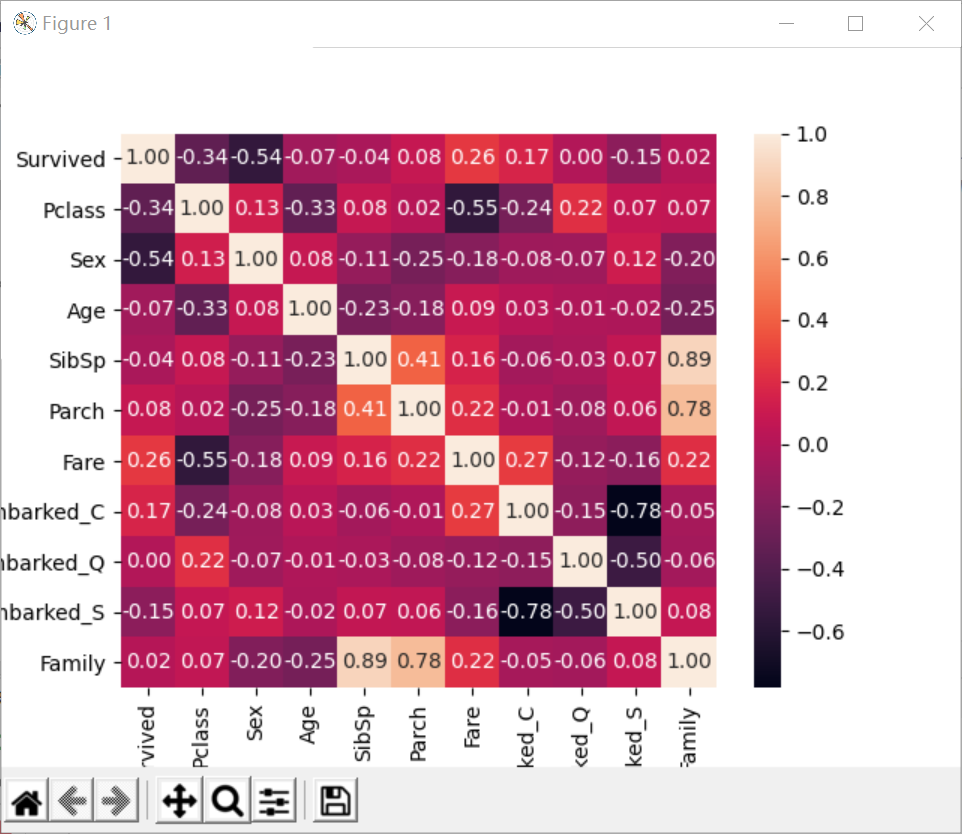
将数据中的Parch和SibSp再加一得出家庭总人数

独热编码：在机器学习算法中，我们经常会遇到分类特征，例如：人的性别有男女，祖国有中国，美国，法国等。  
这些特征值并不是连续的，而是离散的，无序的。通常我们需要对其进行特征数字化。

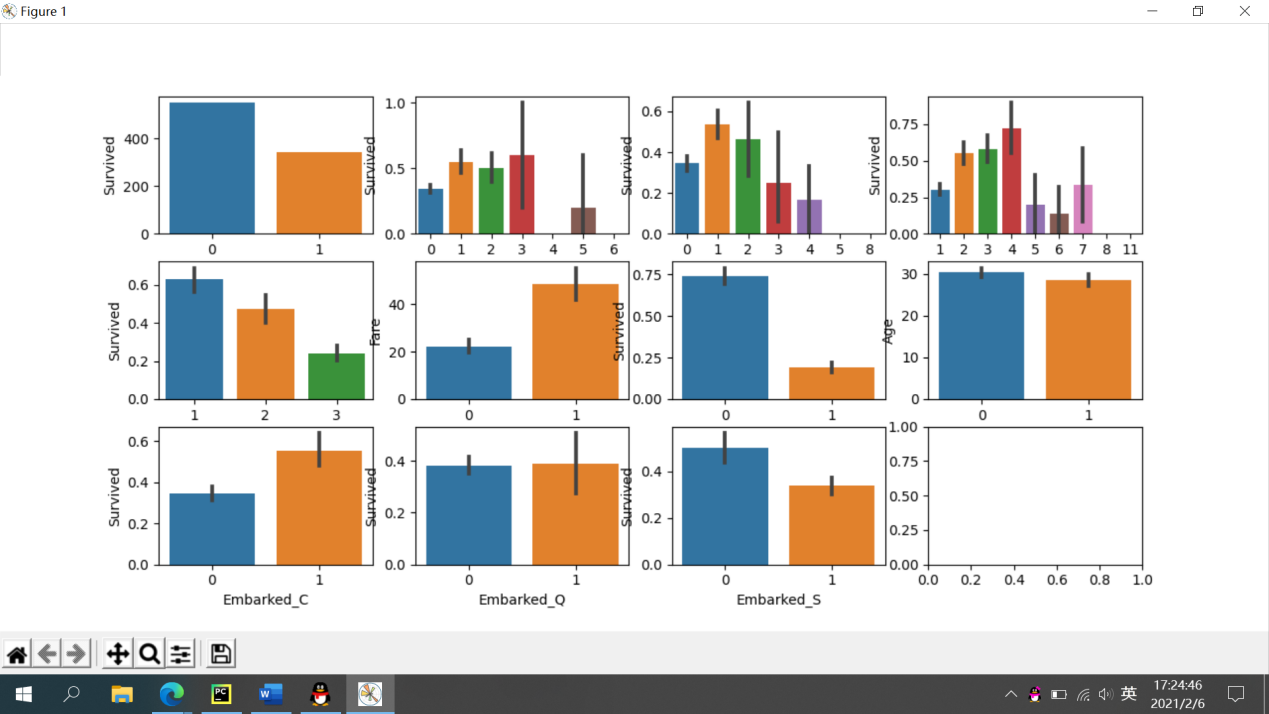
离散特征的编码分为两种情况：1、离散特征的取值之间没有大小的意义，比如color：[red,blue],那么就使用one-hot编码2、离散特征的取值有大小的意义，比如size:[X,XL,XXL],那么就使用数值的映射{X:1,XL:2,XXL:3}

使用pandas中的get\_dummies对Embarked进行热编码

观察数据的相关系数的热力图



2.绘图



（0，0）：遇难者与生存者的统计（0，1）：Parch对生存的影响

（0，2）：SibSp对生存的影响 （0，3）：Family对生存的影响

（1，0）: 票务对生存的影响 （1，1）：票价对生存的影响

（1，2）：性别对生存的影响 （1，3）：年龄对生存的影响

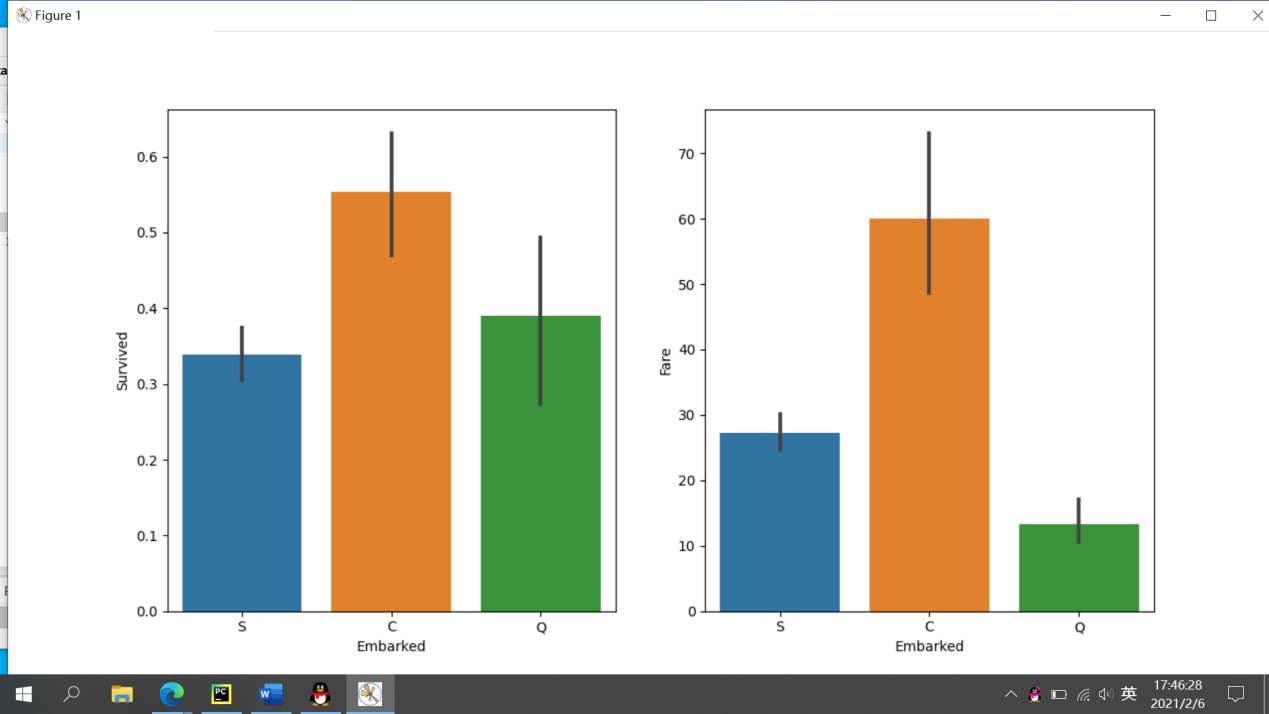
（2，0）：C港口对生存的影响 （2，1）：Q港口对生存的影响

（0，2）：S港口对生存的影响

结论：

遇到者人数远超于获救者，1等票更容易获救，家庭数量过多的，遇害者可能性更大，一等票的人更容易获救，票价高的人更容易获救，女人更容易获救，年龄没有影响，Embarked=C的乘客生还概率高

观察一下港口、票价、生存的关系：



得出C港口的人更容易获救，可能从C港口出发的人都比较有钱，票价高，获救率高

## 3.2 数据建模

泰坦尼克号是一个二分类问题，使用决策树算法进行建模，来对测试数据进行分类预测，参数遍历，在树的深度为8的时候，准确率最高。





在相同的参数情况下，手写算法实现泰坦尼克号分类的准确率更低一点。

# 4、总结

## 4.1 遇到的问题

1.在写决策树的时候，数据划分没有弄清楚，对于连续值的数据划分和离散值的划分的异同点没有搞清楚，导致出现了很大的问题。决策树在数据划分的时候，离散值和连续值最大的区别在于连续值的数据划分不需要删除特征。

2.产生的决策树和调包实现的决策树不一样，原因是因为决策树本身具有随机性，如果不设置随机种子，每次产生的决策树可能都不一样。

3.在回归树模型中，产生的树过于庞大，原因是没有设置最大结点树。在回归树模型中，当特征为0时，返回的不是回归值中最大的，而是回归值的平均值。

4.在构建回归树模型时，因为数据本身的原因，导致mse数据过大，超出范围，解决办法是需要先观察数据，然后将数据归一化处理。

5.在分类树模型中，如果对应特征的特征值只有一种时，在计算信息增益率时，会导致分母为0，解决办法是在当对应特征的特征值数量为1时，需要退出循环。

6.在分类树模型中，预测时由于属性的不同，对应数据的处理方式也不同。

## 4.2 心得体会

通过这次的算法实现，学到了很多，对决策树有了更深一步的理解. 我前面不熟悉啥子的，然后实现过程中，遇到许多困难，然后我查阅资料，问同学等等，逐一解决了问题。发现不能只看视频，还是得动手，才能真正理解其中的关键点和细节。为后续的学习，指明方向，学习不能只停留在表面，得脚踏实地，一步一个脚印。通过这次的算法实现，自己的编程能力，算法理解的深度和以前不一样了。

# 5、附录（代码）

#决策分类树

import pandas as pd  
import math  
import re  
import numpy as np  
  
import random  
from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy\_score  
  
class DecisionTree():  
 def \_\_init\_\_(self,criterion,splitter=**'best'**,  
 max\_depth=100,min\_samples\_spilt=2,min\_samples\_leaf=1,  
 random\_state=0,max\_leaf\_nodes=None,  
 min\_impurity\_decrease=None,temp\_depth=0):  
 self.criterion=criterion  
 self.splitter=splitter  
 self.max\_depth=max\_depth  
 self.min\_samples\_spilt=min\_samples\_spilt  
 self.min\_samples\_leaf=min\_samples\_leaf  
 self.random\_state=random\_state  
 self.max\_leaf\_nodes=max\_leaf\_nodes  
 self.min\_impurity\_decrease=min\_impurity\_decrease  
 self.temp\_depth=temp\_depth  
  
 *# def translate(self,df):  
 # # 将特征的特征值对应转化为数字  
 # dict1 = {'setosa': 0, 'versicolor': 1, 'virginica': 3}  
 # df.loc[:, 'Species'] = df.loc[:, 'Species'].replace(dict1)  
 # return df* def spilt\_data(self,data, feature, feature\_value): *# 根据特征划分数据集（数据集，特征，特征值）* columns = list(data)  
 columns.remove(feature)  
 temp=[]  
 feature = data.columns.get\_loc(feature) *# 特征所对应的index* for i in data.values.tolist():  
 *# 每一行的特征值所对的列表* if i[feature] == feature\_value: *# feature下所对的特征值和分类所对的特征值相同* item = i[:feature] *# 截取特征值的前面* item.extend(i[feature + 1:]) *# 截取特征值后面的部分* temp.append(item) *# 在特征值下分类的数据* return pd.DataFrame(temp, columns=columns)  
  
 def spilt\_data\_continuity(self,data, feature, feature\_value, lr): *# 连续值下的划分数据（不删除数据）* columns = list(data)  
 *# 小于划分点的数据* if lr == **'L'**:  
 item = data  
 item = item[item[feature] <= feature\_value] *# 某个特征下所对应某个特征值的数据* else: *# 大于划分点的数据* item = data  
 item = item[item[feature] > feature\_value] *# 某个特征下所对应某个特征值的数据* return pd.DataFrame(item, columns=columns)  
  
 def calculat\_experience\_hentropy(self,df):  
 category = list([i for i in df.iloc[:, -1]]) *# 类别* num = df.shape[0] *# 数据的数量* category\_num\_count = {}  
 for i in category: *# 每个类别的数量* if i not in category\_num\_count:  
 category\_num\_count[i] = 1  
 else:  
 category\_num\_count[i] += 1  
 if self.criterion != **'gini'**: *# ID.3和C4.5算法* experience\_hentropy = 0  
 for k in category\_num\_count.values():  
 experience\_hentropy += (k / num) \* (math.log(k / num, 2))  
 experience\_hentropy = -experience\_hentropy  
 return experience\_hentropy  
 elif self.criterion == **'gini'**: *# cart算法* gini = 0  
 for k in category\_num\_count.values():  
 gini += (k / num) \*\* 2  
 gini = 1 - gini  
 return gini  
  
 def calculat\_experience\_condition\_hentropy(self,df,label\_property, labels):  
 random.seed(self.random\_state)  
 num = df.shape[0] *# 数据集的数量* feature\_num = df.shape[1] - 1 *# 特征的数量* gain = 0  
 gini = 9999.99  
 feature = None  
 gain\_rate = 0  
 continuity\_feature\_value = None  
 gain\_item\_rate = 0  
 for i in df: *# 循环每一个特征* if i == **'Species'**:  
 break  
 single\_feature\_list = list([k for k in df[i]]) *# 每个特征下的所有特征值* single\_feature\_num = len(list(set(single\_feature\_list))) *# 每个特征下的所有特征值的数量  
 # if single\_feature\_num==1:  
 # break* single\_feature\_num\_count = {}  
 for j in single\_feature\_list: *# 每个特征下每个特征值的数量* if j not in single\_feature\_num\_count:  
 single\_feature\_num\_count[j] = 1  
 else:  
 single\_feature\_num\_count[j] += 1  
 experience\_condition\_hentropy = 0  
 *# condition\_gini=0* continuity\_feature\_value\_item = None *# 连续数据下的最优划分点* if self.criterion == **'entropy'**:  
 idx = labels.index(i)  
 if label\_property[idx] == 0: *# 离散值* for m in single\_feature\_num\_count.keys(): *# 循环每个特征下每个特征值* item = df  
 item = item[item[i] == m] *# 某个特征下所对应某个特征值的数据* experience\_condition\_hentropy -= (-self.calculat\_experience\_hentropy(item)) \* (  
 single\_feature\_num\_count[m] / num) *# 计算经验条件熵* elif label\_property[idx] == 1: *# 连续值* temp = 9999.99  
 single\_feature\_list = list(set([k for k in df[i]])) *# 单个特征下的所有特征值* single\_feature\_list.sort() *# 单个特征下的所有特征值排序* for k in range(len(single\_feature\_list) - 1): *# 二分法* feature\_value = (float(single\_feature\_list[k]) + float(single\_feature\_list[k + 1])) / 2  
 data\_left = self.spilt\_data\_continuity(df, i, feature\_value, **'L'**) *# 小于划分点的数据* data\_right = self.spilt\_data\_continuity(df, i, feature\_value, **'R'**) *# 大于划分点的数据  
 # 计算对应划分点下的经验条件熵* experience\_condition\_hentropy = (len(data\_left) / len(df)) \* \  
 self.calculat\_experience\_hentropy(data\_left) + \  
 (len(data\_right) / len(df)) \* \  
 self.calculat\_experience\_hentropy(data\_right)  
 if experience\_condition\_hentropy <= temp: *# 找出最大的熵值所对应的划分点* temp = experience\_condition\_hentropy  
 continuity\_feature\_value\_item = feature\_value *# 对应的最优划分点* experience\_condition\_hentropy = temp  
 gain\_item = self.calculat\_experience\_hentropy(df) - experience\_condition\_hentropy *# 计算信息增益  
 # print("%s的特征的信息增益为:%.3f" % (i, gain\_item))* if (gain\_item > gain): *# 找出信息增益最大的，作为最优特征* gain = gain\_item  
 feature = i  
 continuity\_feature\_value = continuity\_feature\_value\_item  
 elif self.criterion == **'entropyRate'**:  
 hd = 0  
 idx = labels.index(i)  
 if label\_property[idx] == 0:  
 for m in single\_feature\_num\_count.keys(): *# 循环每个特征下每个特征值* item = df  
 item = item[item[i] == m] *# 某个特征下所对应某个特征值的数据* experience\_condition\_hentropy -= (-self.calculat\_experience\_hentropy(item)) \* (  
 single\_feature\_num\_count[m] / num) *# 计算经验条件熵* hd -= (single\_feature\_num\_count[m] / num) \* math.log((single\_feature\_num\_count[m] / num), 2)  
 gain\_item = self.calculat\_experience\_hentropy(df) - experience\_condition\_hentropy *# 计算信息增益* gain\_item\_rate = gain\_item / hd  
 elif label\_property[idx] == 1:  
 temp = 0  
 single\_feature\_list = list(set([k for k in df[i]]))  
 single\_feature\_list.sort()  
 for k in range(len(single\_feature\_list) - 1):  
 if single\_feature\_num == 1: *# 如果当前特征值只有一个，避免hd为0，直接跳出循环* break  
 feature\_value = (float(single\_feature\_list[k]) + float(single\_feature\_list[k + 1])) / 2  
 data\_left = self.spilt\_data\_continuity(df, i, feature\_value, **'L'**)  
 data\_right = self.spilt\_data\_continuity(df, i, feature\_value, **'R'**)  
 experience\_condition\_hentropy = (len(data\_left) / len(df)) \* \  
 self.calculat\_experience\_hentropy(data\_left) + \  
 (len(data\_right) / len(df)) \* \  
 self.calculat\_experience\_hentropy(data\_right)  
 hd = -((len(data\_left) / num) \* math.log((len(data\_left) / num), 2) +  
 (len(data\_right) / num) \* math.log((len(data\_right) / num), 2))  
 gain\_item = self.calculat\_experience\_hentropy(df) - experience\_condition\_hentropy *# 计算信息增益* gain\_item\_rate = gain\_item / hd  
 if gain\_item\_rate >= temp: *# 找出信息增益率最大的划分点* temp = gain\_item\_rate  
 continuity\_feature\_value\_item = feature\_value  
 gain\_item\_rate = temp  
 *# print("%s的特征的信息增益比为:%.3f"%(i,gain\_item\_rate))* if (gain\_item\_rate >= gain\_rate): *# 找出信息增益比值最大的，作为最优特征* gain\_rate = gain\_item\_rate  
 feature = i  
 continuity\_feature\_value = continuity\_feature\_value\_item  
 *# cart算法* elif self.criterion == **'gini'**:  
 idx = labels.index(i)  
 if label\_property[idx] == 0: *# 离散值* for m in single\_feature\_num\_count.keys(): *# 循环每个特征下每个特征值* item1 = df  
 item1 = item1[item1[i] == m] *# 某个特征下所对应某个特征值的数据* item2 = df  
 item2 = item2[item2[i] != m]  
 gini\_item = (single\_feature\_num\_count[m] / num) \* self.calculat\_experience\_hentropy(item1) \  
 + ((num - single\_feature\_num\_count[m]) / num) \* self.calculat\_experience\_hentropy(item2)  
 *# print("%s的特征下%s的基尼指数为:%.3f" % (i,m, gini\_item))* if (gini\_item <= gini): *# 找出基尼指数最小的，作为最优特征* gini = gini\_item  
 feature = i  
 elif label\_property[idx] == 1: *# 连续值* single\_feature\_list = list(set([k for k in df[i]])) *# 单个特征的特征值* single\_feature\_list.sort() *# 单个特征的特征值排序* for k in range(len(single\_feature\_list) - 1): *# 二分法* feature\_value = (float(single\_feature\_list[k]) + float(single\_feature\_list[k + 1])) / 2  
 data\_left = self.spilt\_data\_continuity(df, i, feature\_value, **'L'**) *# 小于划分点的数据* data\_right = self.spilt\_data\_continuity(df, i, feature\_value, **'R'**) *# 大于划分点的数据  
 # 基尼系数计算* gini\_item = (len(data\_left) / num) \* self.calculat\_experience\_hentropy(data\_left) \  
 + (len(data\_right) / num) \* self.calculat\_experience\_hentropy(data\_right)  
 *# print("%s的特征下%s的基尼指数为:%.3f" % (i, feature\_value, gini\_item))* if (gini\_item <= gini): *# 找出基尼系数最小的，作为最优特征* gini = gini\_item  
 feature = i *# 最优特征* continuity\_feature\_value = feature\_value *# 最优划分点  
 # print(feature,continuity\_feature\_value,gini)* return feature, continuity\_feature\_value  
  
 def classic(self,df, label\_property, labels, df\_test):  
 if self.temp\_depth>=self.max\_depth:  
 return df[**'Species'**].value\_counts().keys()[0]  
 classification = [i for i in df[**'Species'**]] *# 类别列表* if classification.count(classification[0]) == len(classification):  
 return classification[0] *# 如果数据集中所有的实例属于同一类，则将类Ck作为该结点类的标记* if df.shape[1] == 1: *# 如果特征为空，则将数据集中实例数最大的类CK作为节点的类标记返回* return df[**'Species'**].value\_counts().keys()[0]  
 feature, continuity\_feature\_value = self.calculat\_experience\_condition\_hentropy(df, label\_property,  
 labels) *# 得出特征和最优划分点* idx = labels.index(feature) *# 判断是连续值还是离散值* if label\_property[idx] == 0: *# 特征为离散特征* tree = {feature: {}}  
 feature\_values = list(set([k for k in df[feature]])) *# 最优特征下所对应的所有特征值* for i in feature\_values: *# 循环每一个特征值* tree[feature][i] = self.classic(self.spilt\_data(df, feature, i), label\_property, labels, df\_test) *# 递归构建树* else: *# 连续值* feature\_item = str(feature) + **"≤"** + str(continuity\_feature\_value)  
 tree = {feature\_item: {}}  
 *# 小于等于最优划分点* tree[feature\_item][**'是'**] = self.classic(self.spilt\_data\_continuity(df, feature, continuity\_feature\_value, **'L'**),  
 label\_property, labels, df\_test)  
 *# 大于最优划分点* tree[feature\_item][**'否'**] = self.classic(self.spilt\_data\_continuity(df, feature, continuity\_feature\_value, **'R'**),  
 label\_property, labels, df\_test) *# 递归构建树* self.temp\_depth+=1  
 *# if accuracy(df\_test, tree, labels)<=accuracy2(df\_test,df):#如果划分节点后的准确率小于等于划分节点前的准确率  
 # return df['Species'].value\_counts().keys()[0]#当前节点为叶节点，返回类别中最大的* return tree  
  
 def predict(self,tree, feature\_list, data):  
 one\_node = list(tree.keys())[0] *# 第一个节点* item = tree[one\_node] *# 第一个节点所对应的键值* result = None  
 for i in item.keys(): *# 循环节点特征所对应的值* try: *# 离散值* if data[one\_node] == i: *# 如果对应数据的节点特征下的值* if type(item[i]).\_\_name\_\_ == **'dict'**: *# 判断该节点下特征值所对应的键值是否是字典（是否有子节点）* result = self.predict(item[i], feature\_list, data) *# 递归* else:  
 result = item[i]  
 except: *# 连续值* value = **''**.join(re.findall(**r"\d+\.?\d\*"**, one\_node)) *# 最优划分点* one\_node = str(one\_node[:-(len(value) + 1)]) *# 特征* if data[one\_node] <= float(value):  
 *# if np.float64(data[one\_node]).item()<=float(value): #如果对应数据的节点特征下的值小于最优划分点，则为左之* if type(item[**'是'**]).\_\_name\_\_ == **'dict'**: *# 判断该节点下特征值所对应的键值是否是字典（是否有子节点）* result = self.predict(item[**'是'**], feature\_list, data) *# 递归* else:  
 result = item[**'是'**]  
 else:  
 *# # elif np.float64(data[one\_node]).item()>float(value): #如果对应数据的节点特征下的值大于最优划分点，则为右支* if type(item[**'否'**]).\_\_name\_\_ == **'dict'**: *# 判断该节点下特征值所对应的键值是否是字典（是否有子节点）* result = self.predict(item[**'否'**], feature\_list, data) *# 递归* else:  
 result = item[**'否'**]  
 return result  
  
 def accuracy(self,test\_data, tree, feature\_list):  
 errorCnt = 0  
 for i in range(len(test\_data)): *# 变量测试集中每一个测试样本  
 # 判断预测与标签中结果是否一致* if test\_data.iloc[i][**'Species'**] != self.predict(tree, feature\_list, test\_data.iloc[i]):  
 errorCnt += 1  
 *# 返回准确率* return 1 - errorCnt / len(test\_data)  
  
 def accuracy2(self,test\_data, df\_train): *# 判断划分节点前的准确率* errorCnt = 0  
 for i in range(len(test\_data)): *# 变量测试集中每一个测试样本  
 # 判断预测与标签中结果是否一致* if test\_data.iloc[i][**'Species'**] != df\_train[**'Species'**].value\_counts().keys()[0]:  
 errorCnt += 1  
 *# 返回准确率* return 1 - errorCnt / len(test\_data)  
  
iris = load\_iris()  
iris\_feature = iris.data  
iris\_label = iris.target  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(iris\_feature, iris\_label, test\_size=0.3, random\_state=42)  
  
decisionTree\_model = DecisionTreeClassifier()  
*# 模型训练*decisionTree\_model.fit(X\_train,Y\_train)  
predict = decisionTree\_model.predict(X\_test)  
*# 模型评估*print(**"调包实现鸢尾花分类的准确率为："**,str(100\*(accuracy\_score(predict,Y\_test)))+**"%"**)  
  
labels = [**'Sepal.Length'**, **'Sepal.Width'**, **'Petal.Length'**, **'Petal.Width'**]  
labels\_s = [**'Species'**]  
X\_test = pd.DataFrame(X\_test, columns=labels)  
Y\_test = pd.DataFrame(Y\_test, columns=labels\_s)  
df\_test=pd.concat([X\_test, Y\_test], axis=1)  
X\_train = pd.DataFrame(X\_train, columns=labels)  
Y\_train = pd.DataFrame(Y\_train, columns=labels\_s)  
df\_train = pd.concat([X\_train, Y\_train], axis=1)  
model=DecisionTree(criterion=**'gini'**)  
label\_property = [1, 1, 1, 1, 2] *# 0为离散，1为连续值，2为类别  
# print(model.classic(df\_train,label\_property, labels, df\_test))*print(**"手写算法实现鸢尾花分类准确率为："**,str(100\*(model.accuracy(df\_test, model.classic(df\_train,label\_property, labels, df\_test), labels)))+**"%"**)  
  
#决策回归树

import pandas as pd  
import numpy as np  
import re  
from numpy import \*  
from sklearn.datasets import load\_boston  
from sklearn.preprocessing import StandardScaler  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error,r2\_score  
from sklearn.model\_selection import train\_test\_split  
  
class Tree():  
 def \_\_init\_\_(self,max\_depth=100,max\_leaf\_nodes=20,temp\_node=0):  
 self.temp\_node=temp\_node  
 self.max\_leaf\_nodes = max\_leaf\_nodes  
 self.max\_depth=max\_depth  
  
 def spilt\_space(self,df,feature,feature\_point,lr):  
 item = df  
 if lr==**'L'**:  
 item = item[item[feature] <= feature\_point] *# 某个特征下所对应某个特征值的数据* else:  
 item = item[item[feature] > feature\_point]  
 *# item1 = item[item[feature] <= feature\_point]  
 # item = item[~item.index.isin(item1.index)]* return item  
 def cacluate\_mse(self,item1,item2,x):  
 c1 = mean(list([float(k) for k in item1[**'类别'**]]))  
 if item2.empty == True:  
 c2 = 0  
 else:  
 c2 = mean(list([float(k) for k in item2[**'类别'**]]))  
 for m in item1[**'类别'**]:  
 m = float(m)  
 c1 = np.float64(c1).item()  
 x += math.pow((m - c1), 2)  
 for n in item2[**'类别'**]:  
 n = float(n)  
 c2 = np.float64(c2).item()  
 x += math.pow((n - c2), 2)  
 return x,c1,c2  
 def choose\_point(self,df):  
 temp=99999999  
 feature =None  
 best\_feature\_point =0  
 value1=None  
 value2=None  
 for i in df:*#循环所有特征  
 # x=0* if i==**'类别'**:  
 break  
 single\_feature\_list = list(set([k for k in df[i]])) *# 每个特征下的所有特征值* single\_feature\_list.sort()*#特征值排序  
 # point={}  
 # for j in single\_feature\_list: #每个特征下每个特征值的数量  
 # point[j]=0* for j in single\_feature\_list:  
 item1=self.spilt\_space(df,i,j,**'L'**)*#分割空间* item2 = self.spilt\_space(df, i, j, **'R'**) *# 分割空间* x,c1,c2=self.cacluate\_mse(item1,item2,0)*#计算每个切分点的平方误差  
 # print("%s下%f的平方误差为%f",i,j,x)* if (x < temp):*#找出最小的平方误差所对应的特征和切分点* temp = x  
 feature = i  
 best\_feature\_point = j  
 value1=c1  
 value2=c2  
 *# point[j]=x* return feature,best\_feature\_point,value1,value2  
 def classic(self,df,labels):  
 if self.temp\_node>6:  
 return mean(df[**'类别'**])  
 classification=[i for i in df[**'类别'**]] *#类别列表* if classification.count(classification[0])==len(classification):  
 return classification[0] *#如果数据集中所有的实例的值相同，则将该值作为该结点的标记* if df.shape[1]==1: *#如果特征为空，则将数据集中实例数最大的值作为节点的类标记返回  
 # item=df['类别'].value\_counts().keys()* return mean(df[**'类别'**])  
 feature,best\_feature\_point,value1,value2=self.choose\_point(df)*#得出特征和最优划分点,以及节点值* feature\_item=str(feature)+**"≤"**+str(best\_feature\_point)  
 tree = {feature\_item: {}}  
 *#小于等于最优划分点* tree[feature\_item][**'是'**] = self.classic(self.spilt\_space(df,feature,best\_feature\_point,**'L'**),  
 labels)  
 self.temp\_node+=1  
 *# 大于最优划分点* tree[feature\_item][**'否'**] = self.classic(self.spilt\_space(df,feature,best\_feature\_point,**'R'**),  
 labels) *# 递归构建树* self.temp\_node += 1  
 return tree  
 def predict(self,tree, feature\_list,data):  
 one\_node = list(tree.keys())[0]*#第一个节点* item= tree[one\_node] *#第一个节点所对应的键值* result=None  
 value=**''**.join(re.findall(**r'-?\d+\.?\d\*e?-?\d\*?'**, one\_node))  
 *# value = ''.join(re.findall(r"\d+\.?\d\*", one\_node)) # 最优划分点* one\_node = str(one\_node[:-(len(value) + 1)])  
 for i in item.keys(): *#循环节点特征所对应的值* if data[one\_node] <= float(value):  
 *# if np.float64(data[one\_node]).item()<=float(value): #如果对应数据的节点特征下的值小于最优划分点，则为左之* if type(item[**'是'**]).\_\_name\_\_ == **'dict'**:*#判断该节点下特征值所对应的键值是否是字典（是否有子节点）* result=self.predict(item[**'是'**], feature\_list, data)*#递归* else:  
 result=item[**'是'**]  
 else: *#如果对应数据的节点特征下的值大于最优划分点，则为右支* if type(item[**'否'**]).\_\_name\_\_ == **'dict'**:*#判断该节点下特征值所对应的键值是否是字典（是否有子节点）* result=self.predict(item[**'否'**], feature\_list, data)*#递归* else:  
 result=item[**'否'**]  
 return result  
 def accuracy(self,test\_data,tree,feature\_list):  
 mse = 0  
 for i in range(len(test\_data)):*# 变量测试集中每一个测试样本  
 # 判断预测与标签中结果是否一致  
 # print(float(float(self.predict(tree, feature\_list, test\_data.iloc[i]))))* mse+=(float(test\_data.iloc[i][**'类别'**]) -float(self.predict(tree, feature\_list, test\_data.iloc[i])))\*\*2  
  
 *# 返回准确率* return mse / len(test\_data)  
boston = load\_boston()  
boston\_feature = boston.data  
boston\_label = boston.target  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(boston\_feature, boston\_label, test\_size=0.3, random\_state=42)  
std\_x = StandardScaler()  
X\_train = std\_x.fit\_transform(X\_train)  
X\_test = std\_x.transform(X\_test)  
std\_y = StandardScaler()  
Y\_train = std\_y.fit\_transform(Y\_train.reshape(-1, 1))  
Y\_test = std\_y.transform(Y\_test.reshape(-1, 1))  
decisionTree\_model = DecisionTreeRegressor(max\_leaf\_nodes=12,random\_state=0)  
decisionTree\_model.fit(X\_train,Y\_train)  
y\_hat=decisionTree\_model.predict(X\_test)  
*# print(y\_hat)*print(**'波士顿房价调包MSE:'**, mean\_squared\_error(Y\_test, y\_hat))  
*# print(boston.feature\_names)*labels = boston.feature\_names  
labels\_s = [**'类别'**]  
X\_test = pd.DataFrame(X\_test, columns=labels)  
Y\_test = pd.DataFrame(Y\_test, columns=labels\_s)  
df\_test = pd.concat([X\_test, Y\_test], axis=1)  
X\_train = pd.DataFrame(X\_train, columns=labels)  
Y\_train = pd.DataFrame(Y\_train, columns=labels\_s)  
df\_train = pd.concat([X\_train, Y\_train], axis=1)  
xs=list(df\_train)  
model=Tree()  
*#print(model.classic(df\_train,xs))*print(**"手写算法实现波士顿房价mse为：%f"** % (model.accuracy(df\_test, model.classic(df\_train, xs),xs)))