《机器学习基础》实验报告

年级、专业、班级	2019 级计算机科学与技术卓越 02 班	姓名	李燕琴
实验题目	决策树算法实践		
实验时间	2021/11/21	实验地点	DS1422
实验成绩		实验性质	☐验证性 ☐设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

一、实验目的

掌握决策树回归算法原理。

二、实验项目内容

1. 理解并描述决策树分类、回归算法原理。

2. 编程实践，将决策树分类、回归算法分别应用于合适的数据集(如鸢尾花、波士顿房价预测、UCI 数据集、Kaggle 数据集)，要求算法至少用于两个数据集(分类 2 个，回归 2 个)。

三、实验过程或算法（源程序）

(一)决策树原理

“决策树”主要采用“分而治之”的思想。一般的，一棵决策树包含一个根结点、若干个内部结点和若干个叶结点。 其中根结点和内部结点可以称为中间简洁，对应一次属性测试和子结点划分，且根结点包含样本全集，内部结点包含通过祖先结点的属性测试而划分到该结点的样本集；叶结点，对应于一类决策结果（或一个决策值）。

从根结点到每个叶结点的路径，可以理解为一个判定测试序列。具体算法参考图 1。

输入：训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$；
　　属性集 $A=\{a_1,a_2,\ldots,a_d\}$.
过程：函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3: 　将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A=\varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6: 　将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10: 　为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11: 　**if** D_v 为空 **then**
12: 　　将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13: 　**else**
14: 　　以 TreeGenerate(D_v, $A\setminus\{a_*\}$)为分支结点
15: 　**end if**
16: **end for**
输出：以 node 为根结点的一棵决策树

图 1 决策树学习基本算法（源自《机器学习》-周志华）

1、 特殊情况预处理

当划分到某结点时，处于以下集中情况需要视作叶结点处理，其中叶结点值为样本集中数量最多的 y（分类）或样本集 y 的均值（回归）：

（1）样本集个数少于一定阈值（min_samples_leaf）；

（2）样本集中于同一类（分类）或方差较小（回归）；

（3）结点深度超过一定阈值（max_depth）；

（4）最优划分参考值大于一定阈值（criterion_threshold）。

2、 划分选择

决策树学习的关键是如何选择最优划分属性。一般而言，随着划分过程不断进行，决策树的分支结点所包含的样本更大可能属于同一类别，即结点的"纯度"(purity) 越来越高。一般度量样本集合纯度最常用的指标有信息增益（ID3）、信息增益率（C4.5）、基尼指数（CART）。

各个指标应用一般总结如图 2。本次实验主要使用基尼指数和误差平方和，作为分类树和回归树的实现（其他方法应用相似，具体见课本《机器学习》-周志华）。

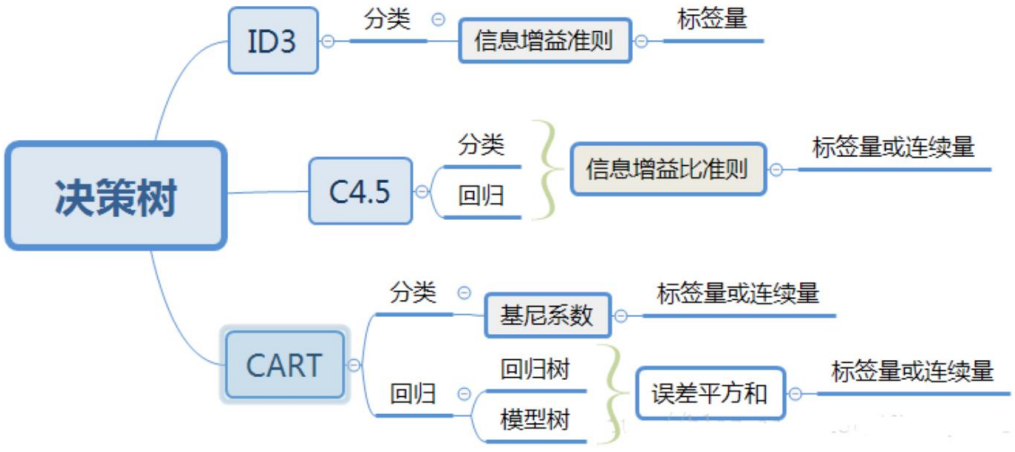

图 2 决策树最优划分属性应用（源自网络）

分类➔基尼指数：

首先计算根据各个类别占比p_k，计算基尼值；再通过基尼值计算基尼指数，选择基尼指数最小的作为最优划分属性。具体计算如下：

$$\begin{aligned}\mathrm{Gini}(D) &= \sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k} p_k p_{k'} \\ &= 1-\sum_{k=1}^{|\mathcal{Y}|} p_k^2 \ .\end{aligned}$$

$$\mathrm{Gini\_index}(D,a) = \sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Gini}(D^v) \ .$$

回归➔误差平方和：

回归主要针对连续值，而对于连续值属性，一般采用连续属性离散化技术。取其所有取值从小到大排序，$\{a^1, a^2, \ldots, a^n\}$，其中取 n-1 个前后值的均值，作为划分候选值。该候选值将原始样本集分为两类，即对应属性值大于等于该候选值的样本（R1），和对应属性值小于该候选值的样本（R2）。其中预期结点值取样本集均值，结点误差平方和计算如下，从众多划分候选值中选择方差最小的值。

$$L(j,s) = \sum_{x_i\in R_1(j,s)}(y_i-\hat{c_1})^2 + \sum_{x_i\in R_2(j,s)}(y_i-\hat{c_2})^2$$

$$\hat{c_1} = \frac{\sum_{x_i\in R_1(j,s)} y_i}{|R_1|}$$

$$\hat{c_2} = \frac{\sum_{x_i\in R_2(j,s)} y_i}{|R_2|}$$

3、 预测

根据上述原理建立好决策树模型后，可以根据需要预测的样本，进行每一个结点的属性测试，当被划分到叶结点时，叶结点的数值即为该样本的预测值。

(二) 模型效果评估

1、 分类

分类主要采用常用的预测类和真实类的均方误差和准确率进行评估。

2、 回归

回归主要采用以下三类进行评估：

（1）平均相对误差

$$\frac{1}{N}\times\frac{|y\_true - y\_pred|}{|y\_true|}$$

（2）均方根误差

$$\frac{1}{N} \times \sqrt{(y\_true - y\_pred)^2}$$

（3）R2

$$R^2 = 1 - \frac{u}{v}$$
$$u = \sum (y\_pred - y\_true)^2$$
$$v = \sum (y\_true - \overline{y\_true})^2$$

(三)算法实现逻辑

如图 3 所示，本实验主要实现决策树基类，通过 python 继承关系，构建分类树和决策树。其中，如图 4 所示，决策树积累可以设置 *min_samples_leaf*、*criterion_threshold*、*max_depth* 等参数以放置决策树过拟合或欠拟合；也提供 load_params()、save_params()等接口，便于模型的加载和保存；提供 plot()接口以供决策树过程显示。具体代码见后。

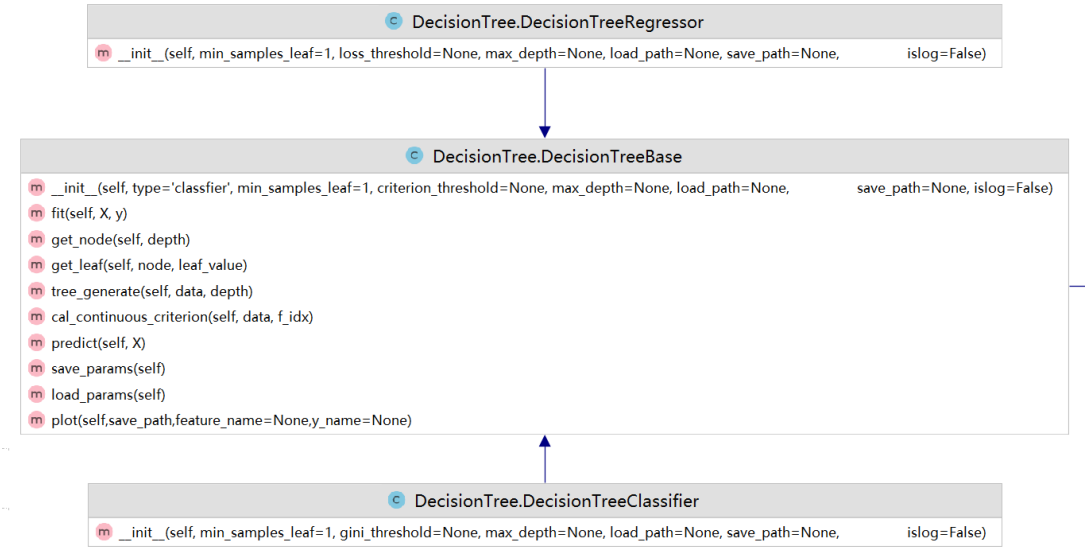

图 3 决策树实现结构

```
def __init__(self, type='classfier', min_samples_leaf=1,
             save_path=None, islog=False):
    '''
    决策树基类初始化
    :param type: ['classfier', 'regressor'] 决策器类别
    :param min_samples_leaf: 最小叶子结点对应样本集数目
    :param criterion_threshold: 最优属性选择阈值
    :param max_depth: 最大的深度
    :param load_path: 参数 load
    :param save_path: 参数 save
    :param islog: 是否开启debug模式
    '''
```

图 4 决策树基类实现

1、 DecisionTree.py

```
# -*- codeing = utf-8 -*-
# @Author: Maxpicca
# @Time: 2021-11-18 19:04
# @Description: DecisionTree

import numpy as np
import os
from collections import Counter
from MLutil import squared_error
import joblib
from pyecharts import options as opts
from pyecharts.charts import Tree


def cal_gini(y):
    ''' 计算基尼值 '''
    y = y.flatten()
    num = len(y)
    y_cnt = Counter(list(y))
    gini = 1 - sum(list(map(lambda k: (y_cnt[k] / num) ** 2, list(y_cnt.keys()))))
    return gini


def cal_gini_index(sub_datalist, n_output, data):
    ''' 计算基尼指数 '''
    father_len = len(data)
    res = map(lambda x: len(x) / father_len * cal_gini(x[:, -n_output]),sub_datalist)
    res = sum(list(res))
```

```
    return res


def cal_squared_error(sub_datalist, n_output, data=None):
    ''' 计算误差平方和，其中 data=None 是为了保证接口一致性 '''
    res = map(lambda x: squared_error(x[:, -n_output], x[:, -
n_output].mean(axis=0)),sub_datalist)
    res = sum(list(res))
    return res


TYPE_LIST = ['classfier', 'regressor']
CRITERION = {'classfier': cal_gini_index, 'regressor': cal_squared_error, }


class DecisionTreeBase:
    ''' 决策树基类实现 '''
    def __init__(self, type='classfier', min_samples_leaf=1,
criterion_threshold=None, max_depth=None, load_path=None,
                 save_path=None, islog=False):
        '''
        决策树基类初始化
        :param type: ['classfier', 'regressor'] 决策器类别
        :param min_samples_leaf: 最小叶子结点对应样本集数目
        :param criterion_threshold: 最优属性选择阈值
        :param max_depth: 最大的深度
        :param load_path: 参数 load
        :param save_path: 参数 save
        :param islog: 是否开启 debug 模式
        '''
        if type not in TYPE_LIST:
            print("输入类型错误")
            return
        self.type = type
        self.criterion_threshold = criterion_threshold
        self.max_depth = max_depth if max_depth is not None else 1000
        self.min_samples_leaf = min_samples_leaf
        self.n_samples = 0
        self.n_features = 0
        self.n_outputs = 0
        self.islog = islog
        self.tree = None
        self.load_path = load_path
        self.save_path = save_path
```

```
def fit(self, X, y):
    # 数据预处理
    if y.ndim == 1:
        y = y.reshape((-1, 1))
    data = np.hstack((X, y))

    # 属性获取
    self.n_samples = len(data)
    self.n_features = X.shape[1]
    self.n_outputs = y.shape[1]

    # 训练过程变量
    self.actual_max_depth = 0
    self.node_num = 0
    self.leaf_num = 0

    if self.load_path is not None:
        # load 参数
        self.tree = self.load_params()
    else:
        # 生成树
        self.tree = self.tree_generate(data, 0)

    if self.save_path is not None:
        # 保存参数
        self.save_params()

def get_node(self, depth):
    ''' 结点创建 '''
    node = {
        'depth': depth,  # 根据函数传递递增
        'is_leaf': False,
        'leaf_value': None,
        'best_fidx': None,
        'best_fvalue': None,
        'sub_node': [],
    }
    self.node_num += 1
    return node

def get_leaf(self, node, leaf_value):
    ''' 叶结点创建 '''
    node['is_leaf'] = True
```

```
        node['leaf_value'] = leaf_value
        self.leaf_num += 1
        return node

    def tree_generate(self, data, depth):
        ''' 以字典的形式，深度优先递归创建树 '''
        if self.islog:
            print("数据长度:%d, 深度:%d\n" % (len(data), depth))

        # 最大高度设置
        self.actual_max_depth = max(self.actual_max_depth, depth)

        # 数据信息获取
        y = data[:, -self.n_outputs]
        leaf_value = 0
        y_cnt = Counter(list(y))

        # 获取叶结点的可能值
        if self.type == TYPE_LIST[0]:  # 分类
            leaf_value = y_cnt.most_common(1)[0][0]
        elif self.type == TYPE_LIST[1]:  # 回归
            leaf_value = y.mean()

        # 生成节点
        node = self.get_node(depth)

        # 样本数量过少
        if len(data) <= self.min_samples_leaf:
            return self.get_leaf(node, leaf_value)

        # data 中的样本属于一类
        if len(y_cnt)==1:
            return self.get_leaf(node, leaf_value)

        # 特征为空，或 data 的样本值都一样，则返回样本数量最多的一类
        # 注：连续值数据，不需要考虑特征为空的情况

        # 深度限制，返回样本数量最多的一类的 label
        if depth >= self.max_depth:
            return self.get_leaf(node, leaf_value)

        # 选择最优划分属性 fidx 和 fvalue
        min_criterion = float('inf')
        min_fvalue = 0
```

```
        min_fidx = 0
        min_sub_datalist = []
        for fidx in range(0, self.n_features):
            fvalue, fgini, f_sub_datalist = self.cal_continuous_criterion(data, fidx)
            if fgini < min_criterion:
                min_criterion = fgini
                min_fvalue = fvalue
                min_fidx = fidx
                min_sub_datalist = f_sub_datalist
        node['best_fidx'] = min_fidx
        node['best_fvalue'] = min_fvalue

        # 最优属性选择阈值判断
        if self.criterion_threshold is not None and min_criterion >
self.criterion_threshold:
            return self.get_leaf(node, leaf_value)

        # 递归子集
        for sub_data in min_sub_datalist:
            if len(sub_data) == 0:
                return self.get_leaf(node, leaf_value)
            else:
                node['sub_node'].append(self.tree_generate(sub_data, depth + 1))

        return node

    def cal_continuous_criterion(self, data, f_idx):
        ''' 计算连续属性值的最优划分度量值 '''
        # 获取所有可能的取值，并从小到达排序
        f_values = list(set(data[:, f_idx]))
        f_values.sort()

        # 开始选择最小的候选值
        min_criterion = float('inf')
        min_fvalue = 0
        min_sub_datalist = []
        for i in range(len(f_values) - 1):
            # 计算后候选值
            f_value = (f_values[i] + f_values[i + 1]) / 2

            # 获取子集
            sub_datalist = []
            data1 = data[data[:, f_idx] >= f_value]
            data2 = data[data[:, f_idx] < f_value]
```

```
            sub_datalist.append(data1)
            sub_datalist.append(data2)

            # 计算最小的基尼指数
            f_criterion = CRITERION[self.type](sub_datalist, self.n_outputs, data)
            if f_criterion < min_criterion:
                min_criterion = f_criterion
                min_fvalue = f_value
                min_sub_datalist = sub_datalist

        return min_fvalue, min_criterion, min_sub_datalist

    def predict(self, X):
        ''' 深度优先递归决策树，进行预测 '''
        def f(node, x):
            if node['is_leaf']:
                return node['leaf_value']
            else:
                fidx = node['best_fidx']
                fvalue = node['best_fvalue']
                sub_id = 0 if x[fidx] >= fvalue else 1
                return f(node['sub_node'][sub_id], x)

        y_pred = np.zeros((len(X), 1))
        for i, r in enumerate(X):
            y_pred[i][0] = f(self.tree, r)
        return y_pred

    def save_params(self):
        ''' 保存参数 '''
        if self.save_path is not None:
            joblib.dump(self.tree, self.save_path)

    def load_params(self):
        ''' 加载参数 '''
        if self.load_path is not None:
            return joblib.load(self.load_path)

    def plot(self,save_path,feature_name=None,y_name=None):
        ''' 绘制决策树 '''
        def f(node):
            data = {}
            if node['is_leaf']:
                data['name'] = y_name[node['leaf_value']] if y_name is not None else
```

```
node['leaf_value']
                return data
            else:
                fidx = node['best_fidx']
                fvalue = node['best_fvalue']
                if feature_name is None:
                    data['name'] = "%d:%.3f"%(fidx,fvalue)
                else:
                    data['name'] = "%s:%.3f"%(feature_name[fidx],fvalue)
                data['children']=[]
                for sub_node in node['sub_node']:
                    data['children'].append(f(sub_node))
                return data

        # 深度优先遍历，确认 children
        data = f(self.tree)
        file_name = os.path.basename(save_path)
        if save_path.split('.')[-1]!='html':
            print("文件名需要以 html 结尾")
            return
        c = (
            Tree()
            .add("",[data],orient='TB') # 从上至下绘制
            .set_global_opts(title_opts=opts.TitleOpts(title=file_name))
            .render(save_path)
        )



class DecisionTreeClassifier(DecisionTreeBase):
    ''' 决策树分类器 '''
    def __init__(self, min_samples_leaf=1, gini_threshold=None, max_depth=None,
load_path=None, save_path=None,
                 islog=False):
        super().__init__(type='classfier', min_samples_leaf=min_samples_leaf,
criterion_threshold=gini_threshold,
            max_depth=max_depth, load_path=load_path, save_path=save_path,
islog=islog, )



class DecisionTreeRegressor(DecisionTreeBase):
    ''' 决策树回归器 '''
    def __init__(self, min_samples_leaf=1, loss_threshold=None, max_depth=None,
load_path=None, save_path=None,
                 islog=False):
```

```
        super().__init__(type='regressor', min_samples_leaf=min_samples_leaf,
criterion_threshold=loss_threshold,
            max_depth=max_depth, load_path=load_path, save_path=save_path,
islog=islog, )
```

2、 test.py

```
# -*- codeing = utf-8 -*-
# @Author: Maxpicca
# @Time: 2021-11-18 11:44
# @Description: test.py

import time

import numpy as np

import DecisionTree
from DecisionTree import DecisionTreeClassifier
from DecisionTree import DecisionTreeRegressor
from MLutil import
get_iris_data,get_wine_data,get_boston_data,get_diabetes_data,
get_airfoil_data
from MLutil import accuracy, mean_squared_loss, mean_squared_root_error,
relative_error, R2
from MLutil import train_test_split

def test(data_name):
    ''' 数据集统一测试 '''
    my_dataset = {
        "iris": {
            "name": "鸢尾花分类数据集",
            "get_fun": get_iris_data,
            "dt":
DecisionTreeClassifier(max_depth=10,save_path="iris_dt.pkl",islog=False),
            'y_name': {
                0:'setosa',
                1:'versicolor',
                2:'virginica',
            },
            'feature_name': None,
            'type':'classfier',
        },
        "wine": {
            "name": "红酒分类数据集",
```

```
            "get_fun": get_wine_data,
            "dt": DecisionTreeClassifier(min_samples_leaf=3,
max_depth=10,islog = False),
            'y_name':None,
            'feature_name': None,
            'type':'classfier',
        },
        'boston':{
            'name':'波士顿回归数据集',
            "get_fun":get_boston_data,
            "dt":DecisionTreeRegressor(min_samples_leaf=5),
            'y_name':None,
            'feature_name':{

0:"CRIM",1:"ZN",2:"INDUS",3:"CHAS",4:"NOX",5:"RM",6:"AGE",7:"DIS",8:"RAD",9
:"TAX",10:"PTRATIO",11:"B",12:"LSTAT",
            },
            'type':'regressor',
        },
        'diabetes':{
            'name':'糖尿病回归数据集',
            "get_fun":get_diabetes_data,
            'dt':DecisionTreeRegressor(min_samples_leaf=30),
            'y_name':None,
            'feature_name': {
                0:'age', 1:'sex', 2:'bmi', 3:'bp', 4:'s1', 5:'s2', 6:'s3',
7:'s4', 8:'s5', 9:'s6'
            },
            'type':'regressor',
        },
        'airfoil':{
            'name':'机翼自噪声回归数据集',
            "get_fun":get_airfoil_data,
            'dt':DecisionTreeRegressor(min_samples_leaf=10),
            'y_name':None,
            'feature_name': {
                # 0:'Frequency', 1:'Angle of attack', 2:'Chord length',
3:'Free-stream velocity', 4:'Suction side displacement thickness'
                0:'freq', 1:'angle', 2:'len', 3:'velocity', 4:'thickness'
            },
            'type':'regressor',
        },
    }
```

```
    print("=" * 30, my_dataset[data_name]["name"], "=" * 30)
    X, Y = my_dataset[data_name]["get_fun"]()
    trainX, testX, trainY, testY = train_test_split(X, Y)

    dt = my_dataset[data_name]["dt"]  # type: DecisionTree.DecisionTreeBase

    start_time = time.time()
    print("模型开始训练")
    dt.fit(trainX, trainY)
    print("模型结构：")
    print("树高度：%d；树节点数目：%d；其中叶节点%d个"%(dt.actual_max_depth,
dt.node_num, dt.leaf_num))
    save_path = "./%s's Decision Tree.html"%(data_name)
    dt.plot(save_path=save_path,
           feature_name=my_dataset[data_name]['feature_name'],
           y_name=my_dataset[data_name]['y_name'],
           )
    print("树结构详见文件",save_path)
    print("模型训练结束，用时%.3fs" % (time.time() - start_time))

    # 预测和评估
    print("测试：")
    predY = dt.predict(testX)

    # 调bug神器
    # print("真实值：",testY.ravel())
    # print("预测值：",predY.ravel())

    if my_dataset[data_name]['type']=='classfier':
        print("损失函数值：%.3f" % (mean_squared_loss(testY, predY)))
        print("预测准确率：%.3f" % (accuracy(testY,predY)))
    elif my_dataset[data_name]['type']=='regressor':
        print("相对误差：%.2f %%" % (relative_error(testY,predY)*100))
        print("均方根误差：%.3f" % (mean_squared_root_error(testY,predY)))
        print("R2:%.3f"%(R2(testY,predY)))

    print()

if __name__ == "__main__":
    test('iris')
    test('wine')
    test('boston')
    test('airfoil')
```

3、 Mtutil.py

```
# -*- codeing = utf-8 -*-
# @Author: Maxpicca
# @Time: 2021-11-19 9:35
# @Description: MLutil 因几次机器学习实验都用到了这些，所以这里写一个MLutil，方便几次实验调用

import numpy as np
import random


# ================================ 常用数据获取函数 ================================

def get_iris_data(filepath='./iris.csv'):
    ''' 获取鸢尾花数据集 '''
    import pandas as pd
    iris_df = pd.read_csv(filepath)
    iris_data = iris_df.values
    X = iris_data[:, :-1]
    Y = iris_data[:, -1][:, np.newaxis]
    return X, Y


def get_wine_data(filepath='./wine.data'):
    ''' 获取红酒数据集 '''
    wine_data = np.loadtxt(filepath, delimiter=",")
    Y = wine_data[:, 0][:, np.newaxis]
    X = wine_data[:, 1:]
    return X, Y


def get_boston_data(file_path='./boston_house_prices.csv'):
    ''' 获取波斯顿房价预测 '''
    import pandas as pd
    boston_df = pd.read_csv(file_path)
    boston_data = boston_df.values
    X = boston_data[1:, :-1].astype(float)
    Y = boston_data[1:, -1][:, np.newaxis].astype(float)
    return X, Y

def get_diabetes_data():
    ''' 获取糖尿病数据集 '''
    from sklearn.datasets import load_diabetes
```

```
    data = load_diabetes()
    X = data.data
    Y = data.target
    return X,Y

def get_airfoil_data(file_path='./airfoil_self_noise.dat'):
    data = np.loadtxt(file_path)
    X = data[:,:-1]
    Y = data[:,-1]
    return X,Y

# ================================ 常用损失函数
================================
def mean_squared_loss(y_true, y_pred):
    y_true = dim_check(y_true)
    y_pred = dim_check(y_pred)
    return ((y_true - y_pred) ** 2).mean(axis=0).sum() / 2


# ================================ 常用评估函数
================================
def dim_check(y):
    if y.ndim ==1:
        y = y.reshape((-1,1))
    return y

def accuracy(y_true, y_pred):
    y_true = dim_check(y_true)
    y_pred = dim_check(y_pred)
    return (y_true == y_pred).mean(axis=0)


def relative_error(y_true, y_pred):
    y_true = dim_check(y_true)
    y_pred = dim_check(y_pred)
    return (np.abs(y_true - y_pred) / np.abs(y_true + 1e-
5)).mean(axis=0).sum()


def squared_error(y_true, y_pred):
    return ((y_true - y_pred) ** 2).sum(axis=0).sum()

def mean_squared_error(y_true, y_pred):
    return ((y_true - y_pred) ** 2).mean(axis=0).sum()
```

```
def mean_squared_root_error(y_true, y_pred):
    y_true = dim_check(y_true)
    y_pred = dim_check(y_pred)
    return np.sqrt(((y_true - y_pred) ** 2).mean(axis=0).sum())

def R2(y_true,y_pred):
    y_true = dim_check(y_true)
    y_pred = dim_check(y_pred)
    u = ((y_pred-y_true)**2).sum()
    v = ((y_true-y_true.mean())**2).sum()
    return 1-u/v

# ================================ 常用激活函数及其求导
================================
def sigmoid(X):
    return 1 / (1 + np.exp(-X))


def sigmoid_diff(y):
    return y * (1 - y)


def tanh(X):
    return (np.exp(X) - np.exp(-X)) / (np.exp(X) + np.exp(-X))


def tanh_diff(y):
    return 1 - y ** 2


def softmax(X):
    return np.exp(X) / np.sum(np.exp(X), axis=1).reshape(-1, 1)  # X / 按照行
求和,得到(n_samples,1)矩阵


# ================================ 其他功能函数
================================
def train_test_split(X, Y, train_percent=0.7, shuffle=True, seed=None):
    ''' 自定义数据分割 '''
    n_smaples = X.shape[0]
    if shuffle:
        idx = np.arange(n_smaples, dtype=int)
        if seed:
```

```
            random.seed(2)
        random.shuffle(idx)
        X = X[idx]
        Y = Y[idx]
    n_train = int(np.floor(n_smaples * train_percent))
    trainX, testX = X[0:n_train], X[n_train:-1]
    trainY, testY = Y[0:n_train], Y[n_train:-1]
    return trainX, testX, trainY, testY


def one_hot_encoder(y, class_encoder=None):
    if class_encoder == None:
        y_set = set(y.ravel())
        class_encoder = {label: idx for idx, label in enumerate(y_set)}
    n_classes = len(class_encoder)
    n_samples = len(y)
    y_one_hot = np.zeros((n_samples, n_classes), dtype=int) + 0.01
    for idx, label in enumerate(y.ravel()):
        y_one_hot[idx, class_encoder[label]] = 1 - 0.01
    return y_one_hot


def one_hot_decoder(y_one_hot, class_decoder=None):
    if class_decoder == None:
        class_decoder = {label: idx for idx, label in
enumerate(range(y_one_hot.shape[1]))}
    y_transfer = y_one_hot.copy()
    for idx, col in enumerate(y_transfer.T):
        # 注意，这里的 col 只是 y_transfer 的一个视图
        col[col == 1] = class_decoder[idx]
    y = np.max(y_transfer, axis=1).astype(int)
    return y.reshape(-1, 1)  # [r,1]
```

4、 实验代码优缺点分析

（1）优点：

- 决策树采用构建基类，继承实现类的方式，代码复用度高；
- 决策树可供设置防止过（或欠）拟合的参数较为灵活，自定义空间大；
- 决策树实现思维导图式可视化，直观；
- 封装了以往机器学习实验的代码，形成了自己的 Mlutil 工具包（至少针对重大的机器学习实验比较受用）；

（2）缺点

- 决策树的剪枝有待实现
- 决策树应用场景具有一定局限性，对于像“糖尿病”等数据集无法做到较好的回归。（听同学讲解，可以设置每个叶结点的数据集方差阈值，以避免对于“糖尿病”数据集出现的过拟合现象。甚妙，报告以记录之。）

5、 参考资料

决策树—分类 - 知乎 (zhihu.com)

决策树—回归 - 知乎 (zhihu.com)

详解决策树、python 实现决策树 - 灰信网（软件开发博客聚合） (freesion.com)

sklearn.tree.DecisionTreeClassifier — scikit-learn 1.0.1 documentation

四、实验结果及分析

(一) 鸢尾花分类数据集

处理结果见图 5，鸢尾花数据集准确率可以达到 0.955，可见模型效果较好。

```
============================== 鸢尾花分类数据集 ==============================
模型开始训练
模型结构：
树高度：5；树节点数目：11；其中叶节点6个
树结构详见文件 ./iris's Decision Tree.html
模型训练结束，用时0.039s
测试：
损失函数值：0.023
预测准确率：0.955
```

图 5 鸢尾花处理结果

iris's Decision Tree.html

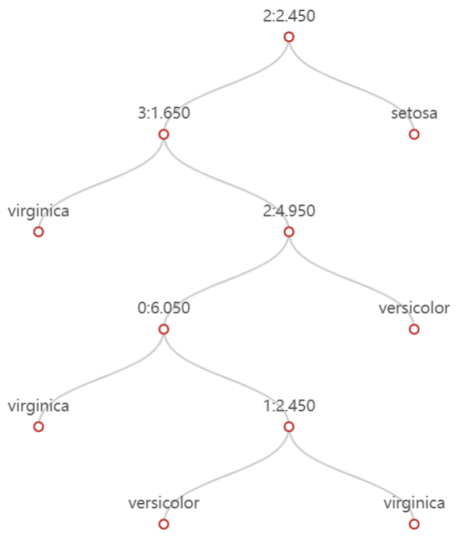

图 6 鸢尾花决策树

(二)红酒分类数据集

处理结果见图 7，红酒数据集准确率可以达到 0.943。

```
============================== 红酒分类数据集 ==============================
模型开始训练
模型结构：
树高度：3；树节点数目：11；其中叶节点6个
树结构详见文件 ./wine's Decision Tree.html
模型训练结束，用时0.132s
测试：
损失函数值：0.028
预测准确率：0.943
```

图 7 红酒处理结果

wine's Decision Tree.html

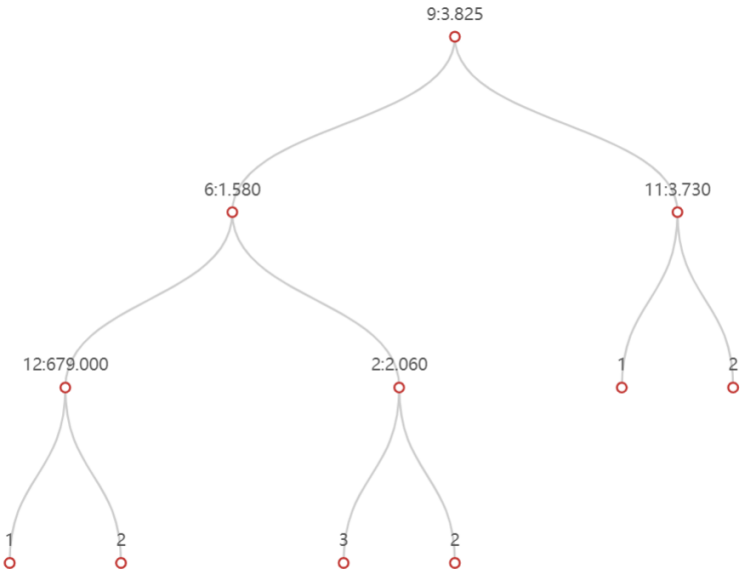

图 8 红酒决策树

(三)波士顿回归数据集

处理结果见图 9，波士顿数据集 R2 可以达到 0.809。除此之外，可见回归树一般会比分类树更高。因为回归数据集的 y 往往更加数据项更多。

```
============================== 波士顿回归数据集 ==============================
模型开始训练
模型结构：
树高度：16；树节点数目：247；其中叶节点124个
树结构详见文件 ./boston's Decision Tree.html
模型训练结束，用时1.909s
测试：
相对误差：16.81 %
均方根误差：4.352
R2:0.809
```

图 9 波士顿处理结果

boston's Decision Tree.html

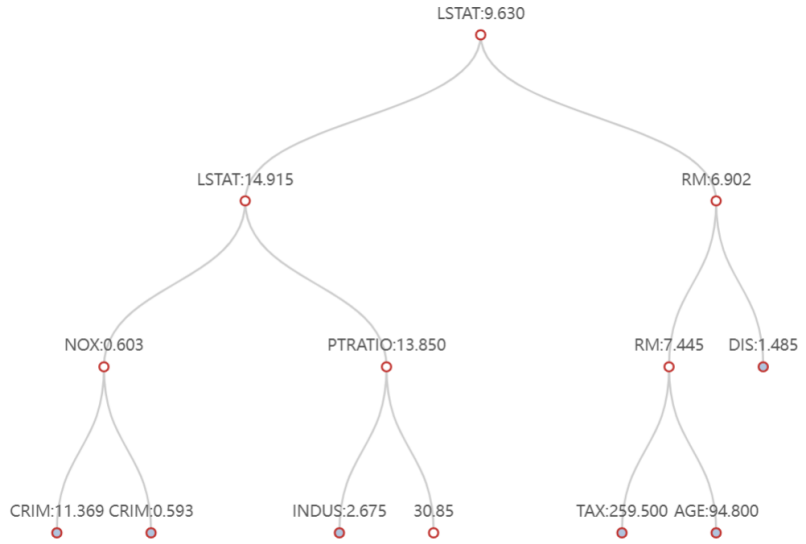

图 10 波士顿决策树

(四)机翼自噪声回归数据集

处理结果见图 11，机翼数据集 R2 可以达到 0.806。

```
============================== 机翼自噪声回归数据集 ==============================
模型开始训练
模型结构：
树高度：16；树节点数目：341；其中叶节点171个
树结构详见文件 ./airfoil's Decision Tree.html
模型训练结束，用时0.262s
测试：
相对误差：1.87 %
均方根误差：3.036
R2:0.806
```

图 11 机翼处理结果

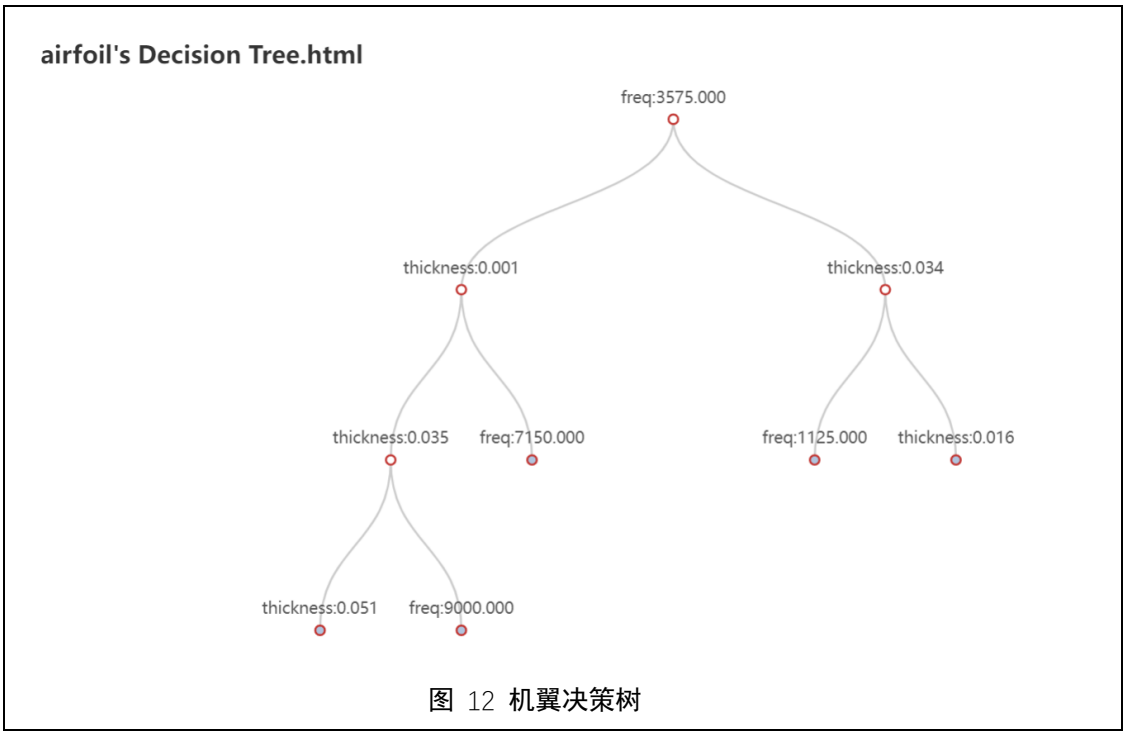
airfoil's Decision Tree.html
freq:3575.000
thickness:0.001
thickness:0.034
thickness:0.035
freq:7150.000
freq:1125.000
thickness:0.016
thickness:0.051
freq:9000.000

图 12 机翼决策树