机器学习实验---决策树

一、实验目的

1. 理解 C4.5 算法原理，能实现 C4.5 算法；
2. 掌握信息增益的计算方式；
3. 掌握决策树的构建和利用决策树进行推断；
4. 理解决策树的优缺点。

二、实验内容

1. 利用 WDBC 数据集，设计一个基于 C4.5 的决策树。以 2/3 的数据为训练集，1/3 为测试集。（可以参考样例，需补充标 XXX 的部分）
   Index of /ml/machine-learning-databases/breast-cancer-wisconsin (uci.edu)

- breast-cancer-wisconsin.data
- breast-cancer-wisconsin.names

2*. 针对数据缺失问题，设计权重划分的决策树。

三、实验报告要求

1. 按实验内容撰写实验过程；
2. 报告中涉及到的代码，每一个模块需要有详细的注释；

四、参考样例

```python
# encoding=utf-8
import matplotlib.pyplot as plt


def create_plot(tree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[],yticks=[])
    create_plot.ax1 = plt.subplot(111, frameon=False, **axprops)
    plot_tree.totalW = float(get_leafnum(tree))
    plot_tree.totalD = float(get_treedepth(tree))
    plot_tree.xOff = -0.5/plot_tree.totalW
    plot_tree.yOff = 1.0
    plot_tree(tree,(0.5,1.0),'')
    plt.show()


def plot_tree(mytree, parent_pt, node_text):
    num_leafs = get_leafnum(mytree)
    depth = get_treedepth(mytree)
    cntr_pt = (plot_tree.xOff + (1.0 + float(num_leafs))/2.0/\
        plot_tree.totalW, plot_tree.yOff)
    plot_mid_text(cntr_pt, parent_pt, node_text)
    plot_node(mytree.attribute, cntr_pt, parent_pt, decision_node)
```

```python
    tree_dict = mytree.dict
    plot_tree.yOff = plot_tree.yOff - 1.0 / plot_tree.totalD
    for key in tree_dict.keys():
        if tree_dict[key].node_type == "internal":
            plot_tree(tree_dict[key], cntr_pt, str(key))
        else:
            plot_tree.xOff = plot_tree.xOff + 1.0 / plot_tree.totalW
            plot_node(tree_dict[key].Class, (plot_tree.xOff, plot_tree.yOff),
                cntr_pt, leaf_node)
            plot_mid_text((plot_tree.xOff, plot_tree.yOff),cntr_pt, str(key))
    plot_tree.yOff = plot_tree.yOff + 1.0/plot_tree.totalD


def plot_node(node_txt, center_pt, parent_pt, node_type):
    create_plot.ax1.annotate(node_txt, xy=parent_pt, \
        xycoords='axes fraction', xytext=center_pt,
        textcoords='axes fraction', va="center", ha="center",\
        bbox=node_type, arrowprops=arrow_args)


def get_leafnum(mytree):
    num_leafs = 0
    tree_dict = mytree.dict
    for key in tree_dict.keys():
        if tree_dict[key].node_type == "internal":
            num_leafs += get_leafnum(tree_dict[key])
        else:
            num_leafs += 1
    return num_leafs


def get_treedepth(mytree):
    max_depth = 0
    tree_dict = mytree.dict
    for key in tree_dict.keys():
        if tree_dict[key].node_type == "internal":
            depth = 1 + get_treedepth(tree_dict[key])
        else:
            depth = 1
        if depth > max_depth:
            max_depth = depth
    return max_depth

def plot_mid_text(cntr_pt, parent_pt, txt):
```

```python
    xMid = (parent_pt[0]-cntr_pt[0])/2.0 + cntr_pt[0]
    yMid = (parent_pt[1]-cntr_pt[1])/2.0 + cntr_pt[1]
    create_plot.ax1.text(xMid, yMid, txt)

import cv2
import time
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from math import log

class Tree(object):
    def __init__(self,node_type,Class = None, attribute = None):
        self.node_type = node_type  # 节点类型（internal 或 leaf）
        self.dict = {} # dict 的键表示属性 Ag 的可能值 ai，值表示根据 ai 得到的子树
        self.Class = Class  # 叶节点表示的类，若是内部节点则为 none
        self.attribute = attribute # 表示当前的树即将由第 attribute 个属性划分（即第 attribute 属性
是使得当前树中信息增益最大的属性）

    def add_tree(self,key,tree):
        self.dict[key] = tree

    def predict(self,attributes):
        #print('attribute', self.attribute)
        if self.node_type == 'leaf' or (attributes[self.attribute] not in self.dict):
            #print('self.Class', self.Class)
            return self.Class

        tree = self.dict.get(attributes[self.attribute])
        return tree.predict(attributes)

# 计算数据集 x 的经验熵 H(x)
def calc_ent(x):
    all_nums = len(x)
    labels = {}
    for label in x:
        if label not in labels.keys():
            labels[label] = 0
        labels[label] += 1
    calc_ent = 0.0
    for key in labels:
        prob = float(labels[key]) / all_nums
        calc_ent -= prob * log(prob, 2)
```

```python
    return calc_ent


# 计算条件熵 H(y/x)
def calc_condition_ent(x, y):
    all_nums = len(x)
    labels_2 = {}
    labels_4 = {}
    labels = {}
    for i in range(all_nums):
        label = x[i]
        y_n = y[i]
        if label not in labels.keys():
            labels[label] = 0
        labels[label] += 1
        if y_n == 2:
            if label not in labels_2.keys():
                labels_2[label] = 0
            labels_2[label] += 1
        if y_n == 4:
            if label not in labels_4.keys():
                labels_4[label] = 0
            labels_4[label] += 1
    calc_condition_ent = 0.0
    for key in labels:
        prob = float(labels[key]) / all_nums
        if key in labels_2.keys() and key in labels_4.keys():
            prob0 = float(labels_2[key]) / float(labels[key])
            prob1 = float(labels_4[key]) / float(labels[key])
            prob2 = -(prob0 * log(prob0, 2) + prob1 * log(prob1, 2))
        else:
            prob2 = 0
        calc_condition_ent += prob * prob2
    return calc_condition_ent
# 计算信息增益
def calc_ent_gain(x,y):
    return calc_ent(x)-calc_condition_ent(x,y)


# C4.5 算法
def recurse_train(train_set,train_label,attributes):


    LEAF = 'leaf'
    INTERNAL = 'internal'


    # 步骤 1——如果训练集 train_set 中的所有实例都属于同一类 C, 则将该类表示为新的叶子节点
```

```python
    label_set = set(train_label)
    if len(label_set) == 1:
        return Tree(LEAF,Class = label_set.pop())

    # 步骤 2——如果属性集为空，表示不能再分了，将剩余样本中样本数最多的一类赋给叶子节点
    class_len = [(i,len(list(filter(lambda x:x==i,train_label)))) for i in label_set] # 计算每一个类出现的个数
    (max_class,max_len) = max(class_len,key = lambda x:x[1])  #出现个数最多的类

    if len(attributes) == 0:
        return Tree(LEAF, Class = max_class)

    # 步骤 3——计算信息增益率,并选择信息增益率最大的属性
    max_attribute = 0
    max_gain_r = 0
    D = train_label
    for attribute in attributes:
        # print(type(train_set))
        A = np.array(train_set[:,attribute].flat) # 选择训练集中的第 attribute 列（即第 attribute 个属性）
        gain = calc_ent_gain(A,D)
        if calc_ent(A) != 0:  ####### 计算信息增益率，这是与 ID3 算法唯一的不同
            gain_r = gain / calc_ent(A)
        if gain_r > max_gain_r:
            max_gain_r,max_attribute = gain_r,attribute
```

```python
    # 步骤 4——如果最大的信息增益率小于阈值,说明所有属性的增益都非常小，那么取样本中最多的类为叶子节点
    if max_gain_r < epsilon:
        return Tree(LEAF,Class = max_class)

    # 步骤 5——依据样本在最大增益率属性的取值，划分非空子集，进而构建树或子树
    sub_attributes = list(filter(lambda x:x!=max_attribute,attributes))
    tree = Tree(INTERNAL,attribute=max_attribute)
    print("[INFO]: create a internal node，feature is %sth"%(max_attribute))
    max_attribute_values = set(train_set[:, max_attribute])
    for value in max_attribute_values:
        indexs = train_set[:, max_attribute] == value
        sub_data = train_set[indexs]
        sub_label = train_label[indexs]
        sub_tree = recurse_train(sub_data, sub_label, sub_attributes)
        tree.add_tree(value, sub_tree)

    return tree
```

```python
def train(train_set,train_label,attributes):
    return recurse_train(train_set,train_label,attributes)

def predict(test_set,tree):
    result = []

    for attributes in test_set:
        tmp_predict = tree.predict(attributes)
        result.append(tmp_predict)

    return np.array(result)

class_num = 2  # wdbc 数据集有 10 种 labels，分别是“2,4”
attribute_len = 9  # wdbc 数据集每个样本有 9 个属性
epsilon = 0.001  # 设定阈值

if __name__ == '__main__':

    print("Start read data...")

    time_1 = time.time()
    raw_data = pd.read_csv('breast-cancer-wisconsin.data', header=None)  # 读取 csv 数据
    data = raw_data.values

    features = data[:, 1:-1]
    # 删除缺失值
    index0 = np.where(features[:,5]!='?')
    features = features[index0].astype('int32')
    labels = data[:,-1][index0]

    # 避免过拟合，采用交叉验证，随机选取 33%数据作为测试集，剩余为训练集
    train_attributes, test_attributes, train_labels, test_labels = train_test_split(features,
labels, test_size=0.33, random_state=0)
    time_2 = time.time()
    print('read data cost %f seconds' % (time_2 - time_1))

    # 通过 C4.5 算法生成决策树
    print('Start training...')
    tree = train(train_attributes,train_labels,list(range(attribute_len)))
    time_3 = time.time()
    print('training cost %f seconds' % (time_3 - time_2))

    print('Start predicting...')
    test_predict = predict(test_attributes,tree)
```

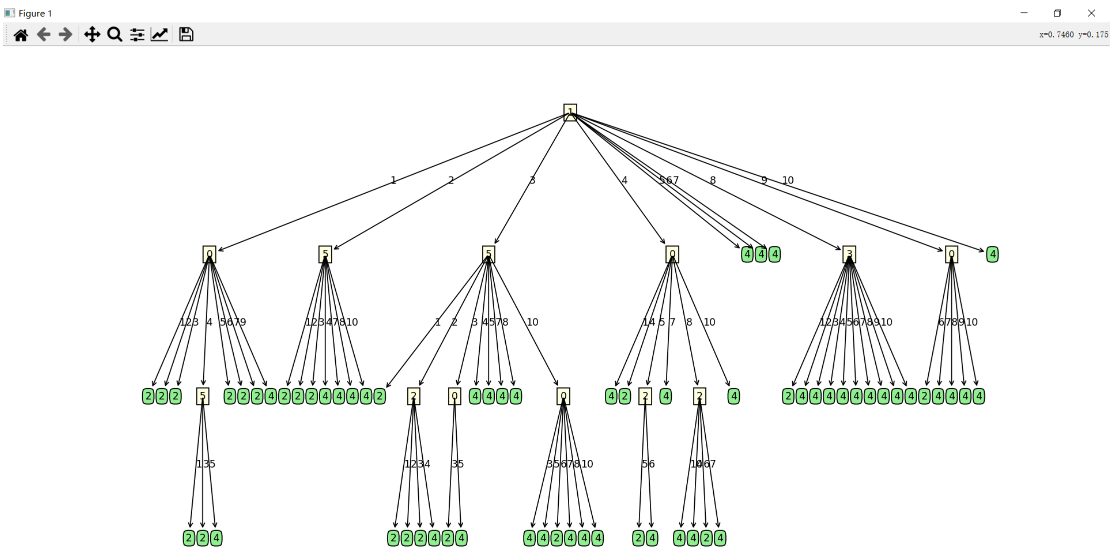
```python
    time_4 = time.time()
    print('predicting cost %f seconds' % (time_4 - time_3))

    print("预测的结果为: ")
    print(test_predict)
    for i in range(len(test_predict)):
        if test_predict[i] == None:
            test_predict[i] = epsilon
    score = accuracy_score(test_labels.astype('int32'), test_predict.astype('int32'))
    print("The accruacy score is %f" % score)
```

五、运行结果

1. 预测测试集上的结果，并输出精度

```
Start predicting...
predicting cost 0.000996 seconds
预测的结果为:
[2 2 4 4 2 2 2 4 2 2 4 2 4 2 2 2 4 4 4 2 2 2 None 2 4 4 2 2 2 4 2 4 4 2 2
 2 4 4 2 4 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 4 None 2 2 2 2 2 2 2 2 2 2 None
 4 2 2 2 2 2 2 4 2 2 2 4 2 4 2 2 4 2 4 4 2 4 2 4 2 4 None 4 4 None 2 2 2 4
 4 2 2 4 None 2 2 4 2 2 4 2 2 2 None 2 2 2 4 2 2 4 4 2 4 2 4 2 2 4 2 2 4 2
 None 2 2 2 2 2 2 2 4 4 2 2 2 4 4 2 2 2 2 4 4 2 4 4 4 4 4 4 2 2 None 2 2 2
 4 None 4 2 2 2 4 2 2 4 4 2 4 2 2 None 4 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 4
 4 2 4 2 4 2 4 2 2 2 2 4 4 2 4 4 4 2 2 4 4]
The accruacy score is 0.915929
```

2*. 绘制出决策树的图



六、实验小结

本次实验是理解 C4.5 算法的原理并实现。理解信息增益在决策树中的意义。构建决策树的过程中，需要利用到递归的思想，每一个内部节点嵌套着一个子树。