# 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
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

```
    # encoding=utf-8

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

#决策树节点定义
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 #将子树添加到当前节点的字典中，用 key 表示子树的属性取值

    def predict(self, attributes):
```

```
        #递归遍历树以进行预测
        #如果当前节点是叶子节点/attributes 中指定的属性值不在字典中
        if self.node_type == 'leaf' or (attributes[self.attribute] not
in self.dict):
            return self.Class #返回当前节点的类别

        return
self.dict[attributes[self.attribute]].predict(attributes) #递归调用子树的
predict 方法，继续预测


# 计算数据集 x 的经验熵 H(x)
def calc_ent(x):
    # Calculate the empirical entropy of the dataset
    x_value_list = set(x) #获取数据集 x 中唯一值的集合
    entropy = 0.0 #初始化熵
    total_samples = len(x) #获取数据集 x 的总样本数

    #遍历数据集 x 中的不同取值
    for x_value in x_value_list:
        p = np.sum(x == x_value) / total_samples #计算 x 中取值为 x_value
的样本在整个数据集中出现的概率
        logp = np.log2(p) #概率的对数
        entropy -= p * logp #更新熵

    return entropy #返回经验熵


# 计算条件熵 H(y/x)
def calc_condition_ent(x, y):
    # Calculate the conditional entropy H(y/x)
    unique_values = np.unique(x) #获取属性 x 的唯一取值
    conditional_entropy = 0 #初始化条件熵

    #遍历属性 x 的不同取值
    for value in unique_values:
        x_indices = np.where(x == value) #获取属性 x 取值为 value 的样本的索
引
        x_subset = y[x_indices] #从 y 中提取对应索引的子集
        conditional_entropy += (len(x_subset) / len(y)) *
calc_ent(x_subset) #计算条件熵的贡献，并累加到总的条件熵中
    return conditional_entropy #返回条件熵

# 计算信息增益
def calc_ent_gain(x, y):
```

```
    # Calculate the information gain
    return calc_ent(y) - calc_condition_ent(x, y) #经验熵 H(y)减去给定 x 条件下的经验条件熵 H(y/x)，即信息增益

# C4.5 算法
#选择具有最大信息增益比的属性，创建内部节点，并根据所选属性的值将数据集拆分为子集，递归构建树
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
    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

    #创建叶子节点，表示剩余样本中最多的类
    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
    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
```

```
    # 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))  #从属性集中移除已选择的最大增益率属性
    tree = Tree(INTERNAL, attribute=max_attribute) #创建内部节点，表示当前树由最大增益率属性划分

    unique_values = np.unique(train_set[:, max_attribute]) #获取最大增益率属性的唯一取值
    for value in unique_values:
        indices = np.where(train_set[:, max_attribute] == value) #获取属性取值等于当前值的样本索引

        sub_train_set, sub_train_label = train_set[indices], train_label[indices] #根据当前属性取值划分子集

        sub_tree = recurse_train(sub_train_set, sub_train_label, sub_attributes) #递归构建树
        tree.add_tree(value, sub_tree) #将子树添加到当前节点的字典中，以当前属性取值为键

    return tree

def train(train_set, train_label, attributes):
    return recurse_train(train_set, train_label, attributes) #调用递归训练函数，开始构建决策树


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

    #对测试集中的每个样本进行预测
    for attributes in test_set:
        #调用决策树的预测方法
        tmp_predict = tree.predict(attributes)
        result.append(tmp_predict)

    return np.array(result) #返回预测结果

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
```

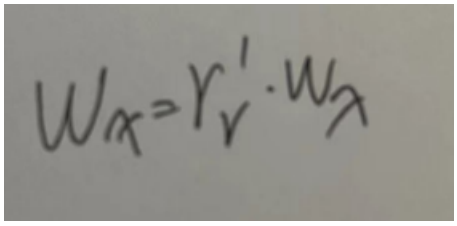
```
epsilon = 0.001  # 设定阈值

if __name__ == '__main__':
    print("Start read data...")

    time_1 = time.time() #开始计时
    #raw_data = pd.read_csv('breast-cancer-wisconsin.data',
header=None)
    #raw_data = pd.read_csv('f:/breast-cancer-wisconsin.data',
header=None, dtype=np.float64)
    raw_data = pd.read_csv('f:/breast-cancer-wisconsin.data',
header=None)

    #获取数据特征和标签
    data = raw_data.values
    features = data[:, 1:-1]
    # 删除缺失值
    data = raw_data.dropna()
    features = data.iloc[:, 1:-1].values
    labels = data.iloc[:, -1].values


    # 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)) #输出读取数据
时间

    # 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)) #输出训练时间

    print('Start predicting...')
    test_predict = predict(test_attributes, tree)
    time_4 = time.time()
    print('predicting cost %f seconds' % (time_4 - time_3)) #输出预测时间

    print(test_predict) #输出预测结果

    #处理可能的 None 值，将其替换为 epsilon
    for i in range(len(test_predict)):
```

```
        if test_predict[i] is None:
            test_predict[i] = epsilon
    score = accuracy_score(test_labels.astype('int32'),
test_predict.astype('int32')) #计算并输出准确率得分
    print("The accuracy score is %f" % score)
```

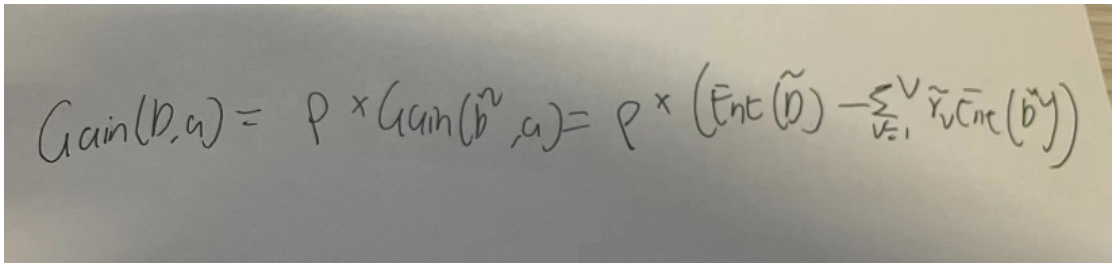
**针对数据缺失问题，设计权重划分的决策树。**

选择信息增益最大的属性作为最优划分属性，那么对于有缺失值的属性，其信息增益就是无缺失值样本所占的比例乘以无缺失值样本子集的信息增益

样本的默认权重为 1，对于无缺失值的样本，划分到子节点时其权重保持不变，缺失样本：



可计算各属性(属性切分点)的最优子集划分方式：



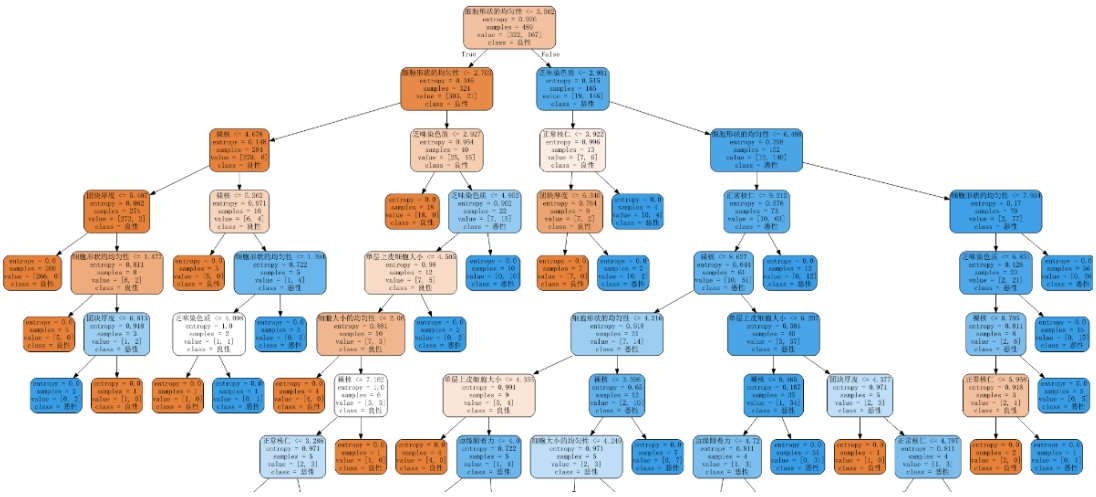
对树的结构做个改进，给每个叶子节点加个数值 N/E，N 表示该叶节点中所包含的总样本数，E 表示与该叶节点的类别不同的样本数，这样就得到样本如有缺失值该如何判断其类别

## 五、运行结果

### 1. 预测测试集上的结果，并输出精度

```
PS C:\Users\11298> & C:/Users/11298/AppData/Local/Programs/Python/Python310/python.exe c:/Users/11298/Desktop/test1.py
Start read data...
read data cost 0.004537 seconds
Start training...
training cost 0.018810 seconds
Start predicting...
predicting cost 0.000000 seconds
[2 2 4 2 4 2 4 2 4 4 2 2 4 4 4 2 2 4 2 2 4 4 2 2 4 4 2 2 4 4 4 2 2 2 2 2 2
 4 4 2 4 4 2 2 2 4 2 2 4 2 4 4 None 2 4 2 2 2 2 2 2 4 2 2 4 4 4 None 2 2 4 2
 2 None 4 2 2 2 2 4 2 2 2 None 2 2 2 4 2 4 4 2 2 2 4 2 2 2 4 2 4 4 2 2 2 4
 2 4 2 2 2 None None 4 2 None 2 2 2 4 4 4 4 2 4 2 2 4 None 4 4 4 2 2 4 4 2
 2 4 2 2 4 4 2 2 2 2 2 4 2 2 2 4 4 4 2 2 2 2 2 2 4 2 None 2 2 2 2 2 2 2
 None 2 2 2 2 4 2 4 4 2 4 4 4 2 2 4 4 2 4 4 4 2 4 2 2 2 2 2 2 2 2 2 4 4 2
 2 None 2 4 2 4 4 4 4 2 2 4 4 2 None 2 2 4 2 2 2 2 2 2 2 4 2]
The accuracy score is 0.878788
PS C:\Users\11298>
```

### 2*. 绘制出决策树的图


细胞形状的均匀性 <= 3.962
entropy = 0.936
samples = 469
value = [322, 667]
class = 良性
True
False
细胞形状的均匀性 <= 2.703
entropy = 0.346
samples = 324
value = [305, 27]
class = 良性
正常染色质 <= 2.981
entropy = 0.515
samples = 165
value = [18, 144]
class = 恶性
核性 <= 4.678
entropy = 0.146
samples = 284
value = [278, 6]
class = 良性
正常染色质 <= 2.907
entropy = 0.954
samples = 40
value = [27, 13]
class = 良性
正常核仁 <= 3.922
entropy = 0.906
samples = 13
value = [7, 6]
class = 良性
细胞形状的均匀性 <= 6.486
entropy = 0.358
samples = 152
value = [13, 140]
class = 恶性
团块厚度 <= 5.000
entropy = 0.062
samples = 275
value = [272, 3]
class = 良性
核性 <= 5.962
entropy = 0.971
samples = 10
value = [6, 4]
class = 良性
entropy = 0.0
samples = 15
value = [18, 0]
class = 良性
正常染色质 <= 4.955
entropy = 0.902
samples = 22
value = [7, 15]
class = 恶性
细胞厚度 <= 5.500
entropy = 0.764
samples = 9
value = [7, 2]
class = 良性
entropy = 0.0
samples = 4
value = [0, 4]
class = 恶性
正常核仁 <= 9.210
entropy = 0.276
samples = 73
value = [10, 63]
class = 恶性
细胞形状的均匀性 <= 7.500
entropy = 0.17
samples = 79
value = [3, 77]
class = 恶性
entropy = 0.0
samples = 266
value = [266, 0]
class = 良性
细胞形状的均匀性 <= 1.417
entropy = 0.811
samples = 9
value = [6, 3]
class = 良性
entropy = 0.0
samples = 5
value = [5, 0]
class = 良性
细胞形状的均匀性 <= 1.394
entropy = 0.722
samples = 5
value = [1, 4]
class = 恶性
单层上皮细胞大小 <= 4.500
entropy = 0.98
samples = 12
value = [7, 5]
class = 良性
entropy = 0.0
samples = 10
value = [0, 10]
class = 恶性
entropy = 0.0
samples = 2
value = [7, 0]
class = 良性
entropy = 0.0
samples = 2
value = [0, 2]
class = 恶性
核性 <= 8.627
entropy = 0.641
samples = 61
value = [10, 51]
class = 恶性
entropy = 0.0
samples = 12
value = [0, 12]
class = 恶性
乏味染色质 <= 4.853
entropy = 0.420
samples = 22
value = [2, 21]
class = 恶性
entropy = 0.0
samples = 56
value = [0, 56]
class = 恶性
entropy = 0.0
samples = 3
value = [3, 0]
class = 良性
团块厚度 <= 6.011
entropy = 0.918
samples = 3
value = [1, 2]
class = 恶性
乏味染色质 <= 4.000
entropy = 1.0
samples = 2
value = [1, 1]
class = 良性
entropy = 0.0
samples = 3
value = [0, 3]
class = 恶性
细胞大小的均匀性 <= 3.00
entropy = 0.881
samples = 10
value = [7, 3]
class = 良性
entropy = 0.0
samples = 2
value = [0, 2]
class = 恶性
细胞形状的均匀性 <= 4.318
entropy = 0.918
samples = 21
value = [7, 14]
class = 恶性
单层上皮细胞大小 <= 6.391
entropy = 0.386
samples = 40
value = [3, 37]
class = 恶性
聚性 <= 8.796
entropy = 0.811
samples = 8
value = [2, 6]
class = 恶性
entropy = 0.0
samples = 15
value = [0, 15]
class = 恶性
entropy = 0.0
samples = 5
value = [5, 0]
class = 良性
团块厚度 <= 6.011
entropy = 0.918
samples = 3
value = [1, 2]
class = 恶性
entropy = 0.0
samples = 2
value = [0, 2]
class = 恶性
entropy = 0.0
samples = 1
value = [1, 0]
class = 良性
entropy = 0.0
samples = 1
value = [1, 0]
class = 良性
entropy = 0.0
samples = 1
value = [0, 1]
class = 恶性
核性 <= 7.162
entropy = 1.6
samples = 6
value = [3, 3]
class = 良性
单层上皮细胞大小 <= 4.333
entropy = 0.991
samples = 9
value = [5, 4]
class = 良性
核性 <= 3.500
entropy = 0.65
samples = 12
value = [2, 10]
class = 恶性
核性 <= 9.063
entropy = 0.195
samples = 35
value = [1, 34]
class = 恶性
团块厚度 <= 4.177
entropy = 0.971
samples = 5
value = [2, 3]
class = 恶性
正常核仁 <= 5.958
entropy = 0.918
samples = 3
value = [2, 1]
class = 良性
entropy = 0.0
samples = 5
value = [0, 5]
class = 恶性
正常核仁 <= 3.286
entropy = 0.971
samples = 5
value = [2, 3]
class = 恶性
entropy = 0.0
samples = 1
value = [1, 0]
class = 良性
entropy = 0.0
samples = 6
value = [1, 0]
class = 良性
细胞新着力 <= 4.0
entropy = 0.722
samples = 3
value = [1, 4]
class = 恶性
细胞大小的均匀性 <= 4.240
entropy = 0.971
samples = 5
value = [2, 3]
class = 恶性
entropy = 0.0
samples = 7
value = [0, 7]
class = 恶性
细胞新着力 <= 4.72
entropy = 0.811
samples = 4
value = [1, 3]
class = 恶性
entropy = 0.0
samples = 31
value = [0, 31]
class = 恶性
entropy = 0.0
samples = 1
value = [1, 0]
class = 良性
正常核仁 <= 4.797
entropy = 0.811
samples = 4
value = [1, 3]
class = 恶性
entropy = 0.0
samples = 2
value = [2, 0]
class = 良性
entropy = 0.0
samples = 1
value = [0, 1]
class = 恶性