山东大学 计算机科学与技术 学院

机器学习与模式识别 课程实验报告

学号：201700301042	姓名： 陈佳睿	班级： 17 人工智能班
实验题目：决策树		
实验学时：2 学时	实验日期： 2019.12.4	

实验目的：

对于决策树在数据集上进行分类任务的实现过程有比较完整和清晰的认识

对于决策树的概念有更加深刻的理解

硬件环境：

Macbook Pro 2017

8G 内存

软件环境：

Pycharm && MatlabR2017a

实验步骤与内容：

实验提供了 csv 格式的文件，是一个关于酒的数据集

这个数据集经常被用来评估分类算法

我们的任务是使用分类算法来评估一个酒的质量是否高于 7

数据集已经对数据进行了标注

酒的得分在[0,6]之间的被映射为 0，得分在 7 分及 7 分以上的被映射为 1

数据一共有 12 列，其中前 11 列是关于酒的相关特性和细节

而最后一列是映射之后的 ground truth label 用于评估这个酒的质量是否达到我们的标准

我们在对于数据进行二元分类的时候只需要使用前 11 列的数据

这里我们使用 ID3 算法来构建决策树

ID3 算法的核心思想是利用信息熵原理选择信息增益最大的属性作为分类属性，

递归地拓展决策树的分枝，完成决策树的构造

而决策树学习本质上是从训练数据集中归纳出一组分类规则

接下来，进行代码实现过程的历程

```
def load_data():
    # 原始数据描述
    # 最上面一行是属性值    一共有4898行数据
    raw_data =[line.strip().split(',') for line in open('/Users/chenjiarui/Desktop/ex7Data/ex7Data.csv')]
    # print(raw_data[1])
    data = raw_data[1:]
    # print(data)
    # 将每一行数据 以list的形式存储起来
    features = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides',
                'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol','quality']
    x_test = data[:800]
    x_train = data[800:]
    return x_train,x_test,features
```

将 csv 文件中的数据值提取出来 并将整体的数据按照接近 6:1 的比例分配测试集和训练集（感觉也没有什么分配硬性的标准，所以随缘分配了）

通过 load_data 函数，返回了训练集、测试集和整个数据集的属性值的列表

Out[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	0
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	0
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	0
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	0
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	0

展示一部分的数据

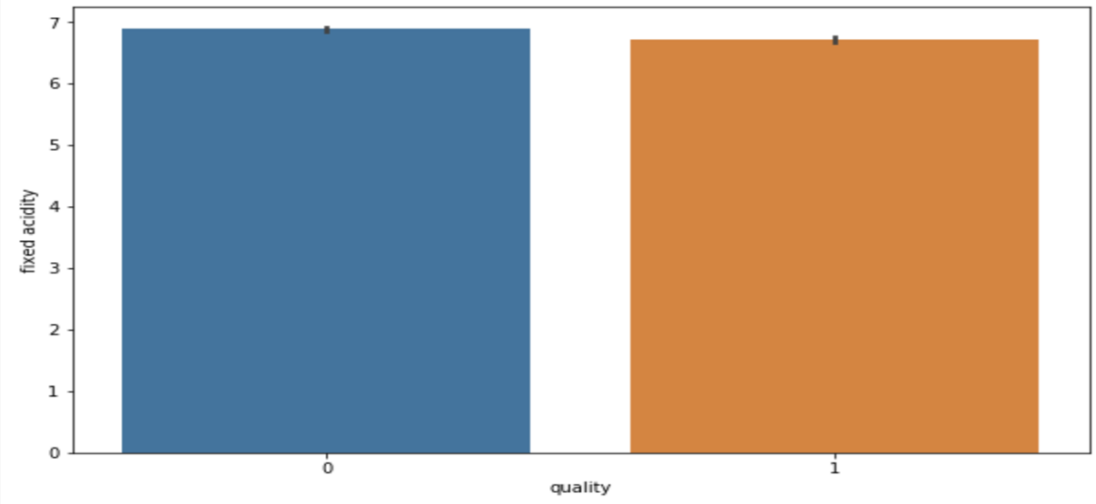
对数据进行简单的统计

In [10]:
```
# main statistics information about each columns
df.describe()
```

Out[10]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	0
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	0
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	0
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	0
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	0
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	1

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b463a02dfd0>
```



对于一个属性的值，我们在 ID3 算法中需要计算其信息增益，所以接下来就是计算整个数据集的熵以及对于每一个属性特征的熵与出现比例的乘积，通过两者相减，得到某一个属性的信息增益，通过打擂法选择出信息增益最大的特征并将其属性作为分类的依据

```python
def choose_best_to_split(data):
    '''
        根据每个特征的信息增益，选择最大的划分数据集的索引特征
    '''
    # 这里一共有11个特征
    count_feature = len(data[0])-1
    # print(count_feature)
    # print(len(data))
    entropy = cal_entropy(data)
    #print(entropy)  # 整个数据集的熵=0.7585780768988335

    max_info_gain = 0.0  # 信息增益最大
    split_fea_index = -1  # 信息增益最大，对应的索引号

    for i in range(count_feature):
        feature_list = [fe_index[i] for fe_index in data]  # 获取该列所有特征值
        # print(feature_list)
        unqval = set(feature_list)  # 去除重复
        Pro_entropy = 0.0  # 特征的熵

        # 遍历属性的所有取值
        for value in unqval:
            sub_data = split_data(data, i, value)
            pro = len(sub_data)/float(len(data))
            Pro_entropy += pro*cal_entropy(sub_data)
            # print(Pro_entropy)

        info_gain = entropy-Pro_entropy
        # print(info_gain)
        if(info_gain > max_info_gain):
            max_info_gain = info_gain
            split_fea_index = i
    return split_fea_index
```

接下来，通过构造决策树生成函数，并不断递归调用该函数来进行节点的生成

```python
def build_decision_tree(dataSet,featnames):
    '''
        字典的键存放节点信息，分支及叶子节点存放值
    '''
    #  读出之前存储的11个特征值
    featname = featnames[:]
    # print(dataSet[0])
    # classlist存储训练集的label值
    classlist = [featvec[-1] for featvec in dataSet]
    # 计算classlist[0] 这个元素在列表中出现的次数
    if classlist.count(classlist[0]) == len(classlist):  # 整个数据集都属于同一个类
        return classlist[0]
    if len(dataSet[0])==1:   # 分完了,没有属性了
        return Vote(classlist)
    # 选择一个最优特征进行划分
    bestFeat = choose_best_to_split(dataSet)
    bestFeatname = featname[bestFeat]
    del(featname[bestFeat])  # 防止下标不准
    DecisionTree = {bestFeatname: {}}
    # 创建分支,先找出所有属性值,即分支数
    allvalue = [vec[bestFeat] for vec in dataSet]
    specvalue = sorted(list(set(allvalue)))  # 使有一定顺序
    for v in specvalue:
        copyfeatname = featname[:]
        DecisionTree[bestFeatname][v] = build_decision_tree(split_data(dataSet, bestFeat, v), copyfeatname)
    return DecisionTree
```

```python
def classify(Tree, featnames, X):
    classLabel=''
    root = list(Tree.keys())[0]
    firstDict = Tree[root]
    featindex = featnames.index(root)  #根节点的属性下标
    #classLabel='0'
    for key in firstDict.keys():   #根属性的取值,取哪个就走往哪颗子树
        if X[featindex] == key:
            if type(firstDict[key]) == type({}):
                classLabel = classify(firstDict[key],featnames,X)
            else:
                classLabel = firstDict[key]
    return classLabel
```

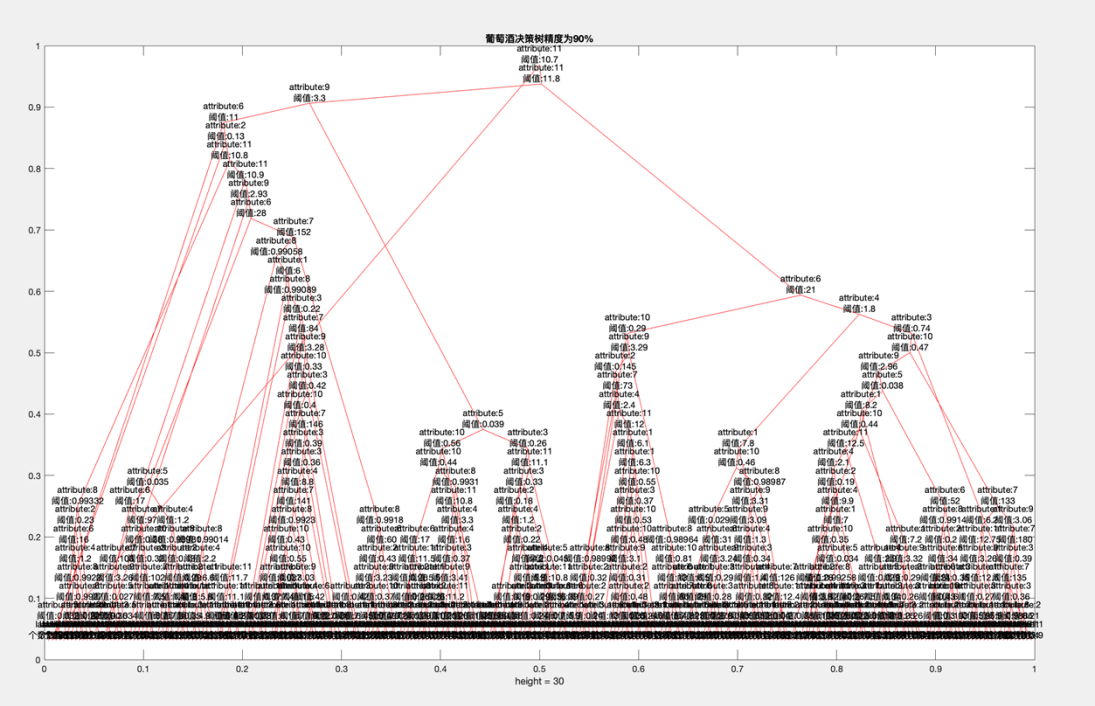
最后得到的对于测试集的准确率在 0.3775

之所以这么低，我认为是因为对于属性的数据值过于离散，导致分类划分过拟合

最终导致决策树的分类效果比较差

接下来是我尝试使用 matlab 来进行这个实验最后绘制出的决策树的样子

最下面的样子因为我的电脑屏幕大小有限看不清楚，只能如此展示



结论分析与体会：

相比直接调用 sklearn 里封装的决策树算法来对 wine 数据集进行二分类，自己手写一个决策树算法要困难地多，并且也要对这个算法的实现细节有更加深刻的认识，通过这个实验，我使用两种不同的编程语言对决策树进行了实现，在 matlab 代码中进行了十折交叉验证，并且使用 matlab 代码绘制了算法所生成的决策树，但是在测试集上的准确度都不高，感觉实验代码还有较大的提升空间。

附录：程序源代码

（1）python 代码

```
from math import log

import operator
```

```
# def loadData():
#     # 原始数据描述
#     # 最上面一行是属性值    一共有 4898 行数据
#     ori_data = pd.read_csv('/Users/chenjiarui/Desktop/ex7Data/ex7Data.csv')
#     # print(ori_data.keys())
#     label = ori_data['quality']
#     # print(type(ori_data))     对于下面这个 DataFrame 对象提取属性列需要使用双重方括号
#     ori_data = ori_data[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
#                 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
#                 'pH', 'sulphates', 'alcohol']]
#     ori_data.to_csv(r'/Users/chenjiarui/Desktop/ex7Data/processData.csv',encoding='gbk')
#     raw_data =[line.strip().split(',') for line in open('/Users/chenjiarui/Desktop/ex7Data/processData.csv')]
#     # print(raw_data[1])
#     data = raw_data[1:]
#     # 将每一行数据 以 list 的形式存储起来
#     # print(type(data[1]))
#     # ans： <class 'str'>
#     features = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides',
#                     'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality']
#     # print(len(data))
#     x_test = data[:800]
#     x_train = data[800:]
```

```
#       return x_train,x_test,features

def load_data():

    # 原始数据描述

    # 最上面一行是属性值     一共有 4898 行数据

    raw_data =[line.strip().split(',') for line in open('/Users/chenjiarui/Desktop/ex7Data/ex7Data.csv')]

    # print(raw_data[1])

    data = raw_data[1:]

    # print(data)

    # 将每一行数据 以 list 的形式存储起来

    features = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides',

                'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol','quality']

    x_test = data[:800]

    x_train = data[800:]

    return x_train,x_test,features



# 计算整个数据集的熵

def cal_entropy(dataSet):

    numEntries = len(dataSet)

    labelCounts = {}

    for featVec in dataSet:

        label = featVec[-1]

        # print(label)

        if label not in labelCounts:
```

```
            labelCounts[label]=0
        labelCounts[label]+=1
    entropy = 0.0
    for key in labelCounts.keys():
        p_i = float(labelCounts[key]/numEntries)
        entropy -= p_i * log(p_i,2)
    return entropy


#  这里因为数据的取值比较特殊 所以不能简单只是根据是否值相同来进行类的划分
def split_data(data,feature_index,value):
    '''
        划分数据集
        feature_index：用于划分特征的列数，例如“年龄”
        value:划分后的属性值：例如“青少年”
    '''
    data_split = []
    for feature in data:
        # if(feature==)
        if feature[feature_index] == value:
            reFeature = feature[:feature_index]
            reFeature.extend(feature[feature_index+1:])
            data_split.append(reFeature)
    return data_split
```

```
def choose_best_to_split(data):
    '''
        根据每个特征的信息增益，选择最大的划分数据集的索引特征
    '''
    # 这里一共有 11 个特征
    count_feature = len(data[0])-1
    # print(count_feature)
    # print(len(data))
    entropy = cal_entropy(data)
    #print(entropy)  # 整个数据集的熵=0.7585780768988335

    max_info_gain = 0.0  # 信息增益最大
    split_fea_index = -1  # 信息增益最大，对应的索引号

    for i in range(count_feature):
        feature_list = [fe_index[i] for fe_index in data]  # 获取该列所有特征值
        # print(feature_list)
        unqval = set(feature_list)  # 去除重复
        Pro_entropy = 0.0  # 特征的熵

        # 遍历属性的所有取值
```

```
            for value in unqval:
                sub_data = split_data(data, i, value)
                pro = len(sub_data)/float(len(data))
                Pro_entropy += pro*cal_entropy(sub_data)
                # print(Pro_entropy)

            info_gain = entropy-Pro_entropy
            # print(info_gain)
            if(info_gain > max_info_gain):
                max_info_gain = info_gain
                split_fea_index = i
        return split_fea_index




def Vote(labels):
    #sorted_label_count[0][0]    次数最多的类标签
    label_count={}
    for label in labels:
        if label not in label_count.keys():
            label_count[label]=0
        else:
            label_count[label]+=1
```

```
        sorted_label_count = sorted(label_count.items(),key = operator.itemgetter(1),reverse = True)
    return sorted_label_count[0][0]




def build_decision_tree(dataSet,featnames):
    '''
        字典的键存放节点信息，分支及叶子节点存放值
    '''
    #   读出之前存储的 11 个特征值
    featname = featnames[:]
    # print(dataSet[0])
    # classlist 存储训练集的 label 值
    classlist = [featvec[-1] for featvec in dataSet]
    # 计算 classlist[0] 这个元素在列表中出现的次数
    if classlist.count(classlist[0]) == len(classlist):   # 整个数据集都属于同一个类
        return classlist[0]
    if len(dataSet[0])==1:    # 分完了,没有属性了
        return Vote(classlist)
    # 选择一个最优特征进行划分
    bestFeat = choose_best_to_split(dataSet)
    bestFeatname = featname[bestFeat]
    del(featname[bestFeat])   # 防止下标不准
```

```
    DecisionTree = {bestFeatname: {}}
    # 创建分支,先找出所有属性值,即分支数
    allvalue = [vec[bestFeat] for vec in dataSet]
    specvalue = sorted(list(set(allvalue)))   # 使有一定顺序
    for v in specvalue:
        copyfeatname = featname[:]
        DecisionTree[bestFeatname][v] = build_decision_tree(split_data(dataSet, bestFeat, v), copyfeatname)
    return DecisionTree



def classify(Tree, featnames, X):
    classLabel=''
    root = list(Tree.keys())[0]
    firstDict = Tree[root]
    featindex = featnames.index(root)   #根节点的属性下标
    #classLabel='0'
    for key in firstDict.keys():    #根属性的取值,取哪个就走往哪颗子树
        if X[featindex] == key:
            if type(firstDict[key]) == type({}):
                classLabel = classify(firstDict[key],featnames,X)
            else:
                classLabel = firstDict[key]
    return classLabel
```

```
if __name__ == '__main__':
    x_train,x_test,features=load_data()
    split_fea_index = choose_best_to_split(x_train)
    # print(split_fea_index)
    newtree = build_decision_tree(x_train,features)
    # print(newtree)

    count = 0
    for test in x_test:
        label = classify(newtree, features, test)

        if (label == test[-1]):
            count = count + 1
    acucy = float(count / len(x_test))
    print(acucy)
```

（2）Matlab 代码：

```
clear all; clc
% 数据处理
f=fopen('/Users/chenjiarui/Desktop/processData.csv');
% 每一行
wine=textscan(f,'%f%f%f%f%f%f%f%f%f%f%f%f%f','delimiter',',');
```

```
fclose(f);
% data=4898*12 （数据规格）
for i=1:12
    data(:,i)=wine{:,i};
end
[m,n]=size(data);

%主程序
%交叉验证
%10 次 10 折交叉验证划分数据集
cov_iter=1;%交叉验证迭代次数
k=10;%划分 10 个子集
while cov_iter<=k
train=data;
train_index=0;
test_index=1;
for i=0:1
    s=sum(data(:,n)==i);
    total=round(s/k);
    sel=randperm(s,total);
    test(test_index:test_index+total-1,:)=data(train_index+sel,:);
    train(train_index+sel,:)=[];
    train_index=train_index+s-total;
    test_index=test_index+total;
end
P=0.9;
tree=decision_tree(train,P);
[right,error]=testtree(tree,test,0,0);
```

```
accuracy(cov_iter)=right/(right+error);
cov_iter=cov_iter+1;
end
display(strcat('测试集平均准确率为',num2str(mean(accuracy)*100),'%'));


%绘制决策树模型
A={};%A 为节点描述
iter=1;%iter 为迭代次数
nodes=0;%根节点序列
[A,~]=print_tree(tree,A,iter,nodes);
[m,n]=size(A);
for i=1:m
    nodes(i)=A{i,n};
    name1{1,i}=A{i,1};
    name2{1,i}=A{i,2};
end
treeplot(nodes);
[x,y]=treelayout(nodes);
for i=1:m
text(x(1,i),y(1,i),{[name1{1,i}];[name2{1,i}]},'VerticalAlignment','bottom','Hor
izontalAlignment','center')
end
title(strcat('葡萄酒决策树精度为',num2str(P*100),'%'));


%计算熵的函数
function [Ent_A,Ent_B] = Ent(data,lamda)
```

```
%data 为按照某一属性排列后的数据集
%lamda 记录按某一属性排序后二分的位置
c=1;   %c 作为 label 类别，一共两个 label（0 和 1）
[m,n]=size(data); %理论来说这里 m=4898 n=12
% disp(m,n)
data1=data(1:lamda,:);
data2=data(lamda+1:end,:);
%将数据分为两个组
Ent_A=0; Ent_B=0;
%计算熵
for i=0:c
    order=i+1;
    data1_p(order)=sum(data1(:,n)==i)/lamda;
    if data1_p(order)~=0
        Ent_A=Ent_A-data1_p(order)*log2(data1_p(order));
    end
    data2_p(order)=sum(data2(:,n)==i)/(m-lamda);
    if data2_p(order)~=0
        Ent_B=Ent_B-data2_p(order)*log2(data2_p(order));
    end
end
end


%寻找最优化分属性 attribute
%最优化分数值 value
function [attribute,value,lamda]=pre(data)
total_attribute=11;  %属性个数
```

```
Ent_parent=0;  %父节点熵
[m,n]=size(data);
% 计算对于二分类任务父节点的总熵值
for j=0:1
    order=j+1;
    data_p(order)=sum(data(:,n)==j)/m;
    if data_p(order)~=0
        Ent_parent=Ent_parent-data_p(order)*log2(data_p(order));
    end
end
%计算最优化分属性
min_Ent=inf;
for i=1:total_attribute
    data=sortrows(data,i);
    for j=1:m-1
        if data(j,n)~=data(j+1,n)
            [Ent_A,Ent_B]=Ent(data,j);
            Ent_pre=Ent_A*j/m+Ent_B*(m-j)/m;
            if Ent_pre<min_Ent
                min_Ent=Ent_pre;
                %记录此时的划分数值和属性
                attribute=i;
                value=data(j,i);
                lamda=j;
            end
        end
    end
    %信息增益
```

```
    gain=Ent_parent-min_Ent;
    if gain<0
        disp('算法有误，请重新设计');
        return
    end
end
end

%建立决策树
%P 控制精确度，防止分支过多，即过拟合现象
function tree=decision_tree(data,P)
[m,n]=size(data);
tree=struct('value','null','left','null','right','null');
[attribute,value,lamda]=pre(data);
data=sortrows(data,attribute);
tree.value=[attribute,value];
tree.left=data(1:lamda,:);
tree.right=data(lamda+1:end,:);
%终止构建条件
for i=0:1
    order= i+1;
    left_label(order)=sum(tree.left(:,n)==i);
    right_label(order)=sum(tree.right(:,n)==i);
end
    [num,max_label]=max(left_label);
    if num~=lamda&&(num/lamda)<P
        tree.left=decision_tree(tree.left,P);
    else
```

```
        tree.left=[max_label num];
    end
    [num,max_label]=max(right_label);
    if num~=(m-lamda)&&(num/(m-lamda))<P
        tree.right=decision_tree(tree.right,P);
    else
        tree.right=[max_label num];
    end
end

%数据可视化，准备绘树状图
function [A,iter]=print_tree(tree,A,iter,nodes)
A{iter,1}=strcat('attribute:',num2str(tree.value(1)));
A{iter,2}=strcat('阈值:',num2str(tree.value(2)));
A{iter,3}=nodes;
iter=iter+1;nodes=iter-1;
if isstruct(tree.left)
    [A,iter]=print_tree(tree.left,A,iter,nodes);
else
    A{iter,1}=strcat('label:',num2str(tree.left(1)));
    A{iter,2}=strcat('个数:',num2str(tree.left(2)));
    A{iter,3}=nodes;
    iter=iter+1;
end
if  isstruct(tree.right)
    [A,iter]=print_tree(tree.right,A,iter,nodes);
else
    A{iter,1}=strcat('label:',num2str(tree.right(1)));
```

```
    A{iter,2}=strcat('个数:',num2str(tree.right(2)));
    A{iter,3}=nodes;
    iter=iter+1;
end
end

%计算测试集正确、错误类别个数
function [right,error]=testtree(tree,test,right,error)
%right 代表测试集中判断正确类别的个数
%error 代表测试集中判断错误类别的个数
attribute=tree.value(1);
value=tree.value(2);
test_left=test(find(test(:,attribute)<=value),:);
test_right=test(find(test(:,attribute)>value),:);
if isstruct(tree.left)
    [right,error]=testtree(tree.left,test_left,right,error);
else
    [m,n]=size(test_left);
    for i=1:m
        if test_left(i,n)==tree.left(1)
            right=right+1;
        else
            error=error+1;
        end
    end
end
if isstruct(tree.right)
    [right,error]=testtree(tree.right,test_right,right,error);
```

```
else
    [m,n]=size(test_right);
    for i=1:m
        if test_right(i,n)==tree.right(1)
            right=right+1;
        else
            error=error+1;
        end
    end
end
end
```