（报告封面）



“数据结构”

课程设计报告

**设计题目**  决策树算法的实现

**姓 名**

**学 号**

**专 业**  计算机科学与技术

**班 级**

**完成日期**  2020.07.05

1. **需求和规格说明**
2. **简介：**决策树是通过一系列规则对数据进行分类的过程。它提供一种在什么条件下会得到什么值的类似规则的方法。它是一个从上到下、分而治之的归纳过程，是决策树的一个经典的构造算法。应用于很多预测的领域，如通过对信用卡客户数据构建分类模型，可预测下一个客户他是否属于优质客户。
3. **分类过程：**分类是数据挖掘、机器学习和模式识别中一个重要的研究领域。数据分类是一个两步过程。第一步，使用已知类别标记的训练数据集建立一个分类模型。例如：图1是一个决策树模型。第二步，对未知标记的数据使用模型进行分类。例如，根据图1的决策树模型，运用自顶而下的属性测试过程，将表2中的样例1-6分别分类为“Y”、“Y”、“Y”、“Y”、“N”、“N”。
4. **算法描述：（核心算法）**

**输入**：训练样例集S，未标记的节点T，属性集A

**输出**：以T为根的决策树

① 如果S中所有样例都是正例，则标记节点T为“Y”，并结束；

② 如果S中所有样例都是反例，则标记节点T为“N”，并结束；

③ 否则，从A中选择一个属性X，(可随机选)标记节点T为X;

④ 设X的所有取值为V1, V2,…,Vn，依据这些取值将S划分为n个子集 S1, S2, …, Sn，建T的n个孩子节点Ti，并分别以Vi作为从T到Ti的分支标号;

⑤ 对每对（Si，Ti，A-{X}），递归调用ID3算法建立一棵以Ti为根的子树；

**天况**

**湿度**

**风况**

***晴***

***多云***

***雨***

***大***

***有***

***无***

**Y**

**Y**

**N**

**N**

**Y**

***正常***

图1. 一个决策树模型的例子

1. **要求：**

① 设计合理的数据结构，编程实现决策树构造算法；

② 给定训练数据集，运用构建的决策树模型，设计合理的文件格式，保存于外存之中；

③ 设计决策树分类算法，根据保存在外存的决策树模型，实现决策树的分类过程，完成对未知类别属性数据样例的分类.

1. **设计**
2. 需求分析：
3. 根据需求①中可知，算法设计的核心是设计决策树算法。根据调查发现决策树算法多用于机器学习方向的应用，其中最常用的算法是ID3和C4.5两种。由算法描述⑤等提示可知主要采用的是ID3算法。
4. 为了完成ID3算法，需要先进行计算香农熵（shanno）用来作为信息熵的参考。ID3算法是在信息增益的基础上进行设计的。
5. 为了调用存在外存的数据集，决定使用文件读写的形式进行训练用数据集和测试用数据集的收集。
6. 设计一定的回显以显示程序实行的进度，并将决策树和程序实现的结果存在文件中。
7. 设计思想：程序结构（如类图），重要的数据结构。主要算法思想（文字描述，不要画框图）

表1：程序函数示意图

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 返回值类型 | 参数表 | 说明 |
| DTA\_createTrainDataset | void | 无 | 创建训练用数据集 |
| DTA\_createTestDataset | void | 无 | 创建训练用数据集 |
| DTA\_calcEntropy | double | vector<vector<string>> &data | 计算信息熵 |
| DTA\_splitDataSet | vector<vector<string>> | vector<vector<string>> data, int axis, string value | 特征划分数据集 |
| DTA\_createFeatureLis | vector<string> | vector<vector<string> > &data, int axis | 创建特征列表 |
| DTA\_chooseBestFeatureToSplit | int | vector<vector<string>> &data | 选择最好的数据集划分方式 |
| DTA\_majorityCnt | string | vector<string> &classList | 返回出现次数最多的分类名称 |
| DTA\_createTree | Node\* | Node\*root,vector< vector<string>>&data,vector<string> &attribute | 递归构建决策树 |
| DTA\_printTree | void | Node \*root, int depth | 打印决策树 |
| DTA\_freeNode | void | Node \*root | 释放节点 |
| DTA\_classify | string | Node \*root, vector<string> &attribute, vector<string> &test | 预测一组数据的结果 |
| DTA\_printResult | void | vector< vector<string> > &data,Node\* root | 打印结果 |

1. 其他设计：

采用了树的数据结构，用来主要储存决策树算法。

其中节点node的结构体定义如下：

//结构体定义

struct Node //决策树节点

{

string attribute; //特征名

string val; //特征值

bool isLeaf; //是否为叶子节点（叶子节点判断终止）

vector<Node\*> childs; //孩子节点

//构造函数，设定初始值

Node()

{

val = ""; //初始特征值

attribute = ""; //初始特征名

isLeaf = false; //默认不是叶子节点

}

};

同时为了实现多个函数之间的互通，和main函数的引用方便，程序同时设计了一些公共变量如下。

//全局变量

vector**<** vector**<**string**>** **>** trainData**;** //创建训练数据集

vector**<** vector**<**string**>** **>** testData**;** //创建实验数据集

vector**<** vector**<**string**>** **>** update**;** //创建缓存更新数据集--//用于创建决策树

vector**<**string**>** attributes**;** //创建特征

Node **\***root **=** **NULL;** //创建决策树的根节点

1. 实现注释：各项要求的实现程度、在完成基本要求的基础上还实现了什么功能？
   1. 实现构造决策树算法，并额外增加打印决策树，便于观察。
   2. 为给定训练数据集设计合理的文件格式，保存于外存之中。同时规范了所有数据集的格式，并添加了程序后端读取的显示。
   3. 使用vector动态数组完成对未知类别属性数据样例的分类，实现了功能，同时也满足兼容不同类型的数据集。
   4. 检验部分并不够完善，还存在部分情况并未包含在程序中。
   5. 数据输出过程中由于使用了vector数组，偶尔会出现动态数组一场的情况。
2. **用户手册**

请严格按照以下要求进行操作：

1. 将训练用数据集按照规定的方式进行整理。（参见用户手册附录1），将数据集存到文件“train.data”中；
2. 将测试用数据集按照规定的方式进行整理。（参见用户手册附录1），

将数据集存到文件“test.data”中；

1. 运行程序，选择运行并等待显示效果。
2. 决策树的结构将在命令行和文件“DecisionTree.txt”文件中打印。
3. 实验数据的结果将会在文件“result.txt”文件中打印。
4. 程序运行过程中出现的回显的含义以及问题如何解决参考用户手册附录2。

用户手册附录：

**附录1：数据集存储的格式：**

训练用数据集文件存储的内容为特征（feature）以及判断结果（result）。测试用数据集存储的内容没有判断结果（result）。

其中文件第一行为表头，应存入特征名（attribute）和结果标头（result）。之后每一行有多个信息元素，属于一条数据。各个信息元素之间用制表符隔开，即tab键。各条数据之间则用回车隔开（以上规则适用于表头元素）。

数据集文件应该使用规定的命名方式，以.data的后缀名结尾。命名需要满足训练用（train）和测试用（test）的名称。具体的情况可以参考附录文件中的数据集文件。

**附录2：程序运行中的问题及解决**

1. 出现回显“XXX is completed”，说明某一步操作已经顺利完成。
2. 出现回显“XXX is not found”,说明系统为寻找到相应的数据文件，此时应该检查文件格式或者文件命名是否出错。
3. 出现回显“XXX is happened”，说明程序运行过程中发生同名错误，请重试。
4. **调试及测试**
5. **测试使用的数据集**

表1. 一个训练数据集

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 天况 | 温度 | 湿度 | 风况 | 分类 |
| 1 | 晴 | 热 | 大 | 无 | N |
| 2 | 晴 | 热 | 大 | 有 | N |
| 3 | 多云 | 热 | 大 | 无 | Y |
| 4 | 雨 | 中 | 大 | 无 | Y |
| 5 | 雨 | 冷 | 正常 | 无 | Y |
| 6 | 雨 | 冷 | 正常 | 有 | N |
| 7 | 多云 | 冷 | 正常 | 有 | Y |
| 8 | 晴 | 中 | 大 | 无 | N |
| 9 | 晴 | 冷 | 正常 | 无 | Y |
| 10 | 雨 | 中 | 正常 | 无 | Y |
| 11 | 晴 | 中 | 正常 | 有 | Y |
| 12 | 多云 | 中 | 大 | 有 | Y |
| 13 | 多云 | 热 | 正常 | 无 | Y |
| 14 | 雨 | 中 | 大 | 有 | N |

表2. 一个待分类的数据集

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 天况 | 温度 | 湿度 | 风况 | 分类 |
| 1 | 晴 | 热 | 正常 | 无 | ？ |
| 2 | 晴 | 热 | 正常 | 有 | ？ |
| 3 | 雨 | 热 | 正常 | 无 | ？ |
| 4 | 晴 | 中 | 正常 | 无 | ？ |
| 5 | 晴 | 冷 | 大 | 有 | ？ |
| 6 | 晴 | 冷 | 大 | 无 | ？ |

1. **测试步骤**
   1. **初步测试**

采用一组数据进行判别

晴 热 大 无 N

* 1. **正式测试**

采用正式的数据集进行测试

晴 热 正常 无

晴 热 正常 有

雨 热 正常 无

晴 中 正常 无

晴 冷 大 有

晴 冷 大 无

1. **运行实例：**

**注：采用的实例为上述文件中的数据。**

天况 温度 湿度 风况 分类

晴 热 大 无 N

晴 热 大 有 N

多 热 大 无 Y

雨 中 大 无 Y

雨 冷 正常 无 Y

雨 冷 正常 有 N

多云 冷 正常 有 Y

晴 中 大 无 N

晴 冷 正常 无 Y

雨 中 正常 无 Y

晴 中 正常 有 Y

多云 中 大 有 Y

多云 热 正 无 Y

雨 中 大 有 N

DTA\_createTrainDataset is completed

晴 热 正常 无

晴 热 正常 有

雨 热 正常 无

晴 中 正常 无

晴 冷 大 有

晴 冷 大 无

DTA\_createTestDataset is completed

DTA\_createTree is completed

DTA\_printTree is completed

DTA\_printResult is completed

DTA\_freeNode is completed

**（六）仍需进一步改进**

1. 读取文件时每一行第一位数字未被读取，可以改进。

2. 输出的文件名字固定，数据集的文件名固定，不能进行改变，可以更换成需要用户输入的方式进行打开文件，这一点可以通过创建文件名的参数做到。

3. 关于程序会出现的问题，进行的反馈和判断并不足够。因为诸多原因，关于文件的判别并不够严谨。

**（七）心得体会**

（可描述课程设计过程中出现的主要问题、原因及解决方法，课程设计的主要收获等）

**出现问题：**

**出现问题1：如何实现决策树算法？**

答：在ID3算法的理解上，我通过计算香侬熵的方式，计算信息增益，从而根据特征来构建决策树。决策树的算法从本质上是一种根据现有的信息所提供的熵值和权重进行决策判断，从而实现经过训练后，可以对相同类型的数据进行决策判断。于是我就将算法的核心部分拆分来实现。这样可以看出，函数式的C++代码更适合本次的设计。

**出现问题2：如何实现对任何类型的数据都可以构建？**

答：这里就涉及到了如何接受任何大小的二维数组。于是我从原来的固定二维数组改为了双重vector数组。这样通过push\_back函数就可以定制动态数组。同时也可以通过迭代器iterator对动态数组进行遍历。这样既没有修改决策树算法的核心，也提高了代码的泛用性。

**出现问题3：如何克服和使用文件读写？**

答：这里原先采用的是file指针读取，但是因为指针读取的复杂性，根据查询的资料，改用了ifstream和ofstream的流式读写方式，在一定程度上降低了程序的复杂性，提高了程序的可读性。

**出现问题4：关于数据集如何处理的问题？**

答：在解决了前面的算法核心问题后，如何处理从数据集获取的数据成了另一个难题。首先是数据划分问题，在这方面我考虑到了spilt函数，但是C++中spilt函数需要重构，我选择了另一种直接判断的方法进行设计，这里希望我未来有时间后可以修改判断的方法。另一个方面就是数据的选择问题，我通过书写判断逻辑进行筛选数据，最终读取数据集的处理函数完成。

**出现问题5：如何进行测试？**

答：这里采用了简单直接的输入数据进行测试的方法。在程序语法阶段，是通过大量的debug和警告提示进行修改，而在程序语法通过后，程序逐渐改为了选择输入数据进行逻辑的测试，当然也有简单的增加回显以提示项目的进程。

**项目收获：**

1. 清楚了解和基本掌握了决策树算法，对数据的处理和录入有了更加深入的了解
2. 对树、顺序表、链表等数据结构有了更极深刻的了解，同时也学习了vector动态数组的用法，对不同的结构应该采用何种算法有了一些认识。
3. 对程序的debug和修改优化，还有如何书写注释，进行了学习，在之后的程序设计过程中会更加有经验。
4. 对C++函数式的设计更加熟悉，对C++语言的一些语法常识有了了解。
5. 完善了从问题出发，再到如何解决问题的具体流程，丰富了设计的经验。

**（八）对课程设计的建议**

希望可以完善一些时间安排，题目方面的描述很棒，但是设计的具体流程需要自己安排。这将耗费大量时间。

**（九）附录⎯⎯源程序**

也可以参见实验附录程序。

//引用头文件

#include<iostream>

#include<string.h>

#include<string>

#include<map>

#include<list>

#include <vector>

#include <set>

#include <algorithm>

#include <map>

#include <math.h>

#include <fstream>

//使用命名空间std

**using** **namespace** std**;**

//宏定义

#define Feature 100 //特征数（feature）+1的上限

#define Sample 7000 //样本数量的 上限

#define FeatureChoice 50 //属性 取值种类 的上限

//结构体定义

struct Node //决策树节点

**{**

string attribute**;** //特征名

string val**;** //特征值

bool isLeaf**;** //是否为叶子节点（叶子节点判断终止）

vector**<**Node**\*>** childs**;** //孩子节点

//构造函数，设定初始值

Node**()**

**{**

val **=** ""**;** //初始特征值

attribute **=** ""**;** //初始特征名

isLeaf **=** **false;** //默认不是叶子节点

**}**

**};**

//全局变量

vector**<** vector**<**string**>** **>** trainData**;** //创建训练数据集

vector**<** vector**<**string**>** **>** testData**;** //创建实验数据集

vector**<** vector**<**string**>** **>** update**;** //创建缓存更新数据集--用于创建决策树

vector**<**string**>** attributes**;** //创建特征

Node **\***root **=** **NULL;** //创建决策树的根节点

//全局函数

void DTA\_createTrainDataset**();** //从文本文件"train.data"读取文本信息，然后创建训练用数据集

void DTA\_createTestDataset**();** //从文本文件"test.data"读取文本信息，然后创建训练用数据集

double DTA\_calcEntropy**(**vector**<** vector**<**string**>** **>** **&**data**);** //计算给定数据集的信息熵

vector**<** vector**<**string**>** **>** DTA\_splitDataSet**(**vector**<** vector**<**string**>** **>** data**,** int axis**,** string value**);** //按照给定特征划分数据集，新的数据集的维度少了一个/axis ：特征下标/value：特征值

vector**<**string**>** DTA\_createFeatureList**(**vector**<** vector**<**string**>** **>** **&**data**,** int axis**);** //创建特征列表

int DTA\_chooseBestFeatureToSplit**(**vector**<** vector**<**string**>** **>** **&**data**);** //选择最好的数据集划分方式

string DTA\_majorityCnt**(**vector**<**string**>** **&**classList**);** //返回出现次数最多的分类名称/如果类标签依不唯一，采用多数表决的方法定义叶子节点的分类

Node**\*** DTA\_createTree**(**Node **\***root**,** vector**<** vector**<**string**>** **>** **&**data**,** vector**<**string**>** **&**attribute**);** //递归构建决策树

void DTA\_printTree**(**Node **\***root**,** int depth**);** //打印决策树到“Tree.txt”中

void DTA\_freeNode**(**Node **\***root**);** //释放节点

string DTA\_classify**(**Node **\***root**,** vector**<**string**>** **&**attribute**,** vector**<**string**>** **&**test**);** //预测一组数据的结果

void DTA\_printResult**(**vector**<** vector**<**string**>** **>** **&**data**,**Node**\*** root**);** //打印结果到“result.txt"中 //将结果打印到“result.txt”中

//函数实现

//从文本文件"train.data"读取文本信息，然后创建训练用数据集

void DTA\_createTrainDataset**()**

**{**

int i**,**j**;** //设置循环变量

int num**;**

char p**,\***p2**;**

string line**;**

string word**;**

vector**<**string**>** temp\_line**;**

ifstream fin**(**"train.data"**);**

//首先获取第一行作为属性

getline**(**fin**,**line**);**

//分割成单词

**for** **(**string**::**size\_type i **=** 0**;**i **<** line**.**size**();** i**++)**

**{**

**if(**line**[**i**]==**'\t'**)**

**{**

**if(!**word**.**empty**())**

**{**

temp\_line**.**push\_back**(**word**);**

word **=** ""**;**

**}**

**}**

**else**

word **+=** line**[**i**];**

**}**

temp\_line**.**push\_back**(**word**);**

word **=** ""**;**

**for(**i **=** 0**;** i **<**temp\_line**.**size**();** **++**i**)**

**{**

attributes**.**push\_back**(**temp\_line**[**i**]);**

**}**

temp\_line**.**clear**();**

**while** **(**fin**.**get**(**p**))**

**{**

getline**(**fin**,**line**);**

// string::size\_type

**for** **(**i **=** 0**;**i **<** line**.**size**();** i**++)**

**{**

**if(**line**[**i**]==**'\t'**)**

**{**

**if(!**word**.**empty**())**

**{**

temp\_line**.**push\_back**(**word**);**

word **=** ""**;**

**}**

**}**

**else**

word **+=** line**[**i**];**

**}**

temp\_line**.**push\_back**(**word**);**

word **=** ""**;**

trainData**.**push\_back**(**temp\_line**);**

temp\_line**.**clear**();**

**}**

fin**.**close**();** //所以需要将最后一行的数据再填入到二维动态数组中

**for(**i **=** 0**;** i **<**attributes**.**size**();** **++**i**)**

**{**

cout **<<** attributes**[**i**]** **<<** "\t"**;**

**}**

cout **<<** endl**;**

**for** **(**i **=** 0**;**i **<** trainData**.**size**();** **++**i**)**

**{**

**for** **(**j **=** 0**;**j **<** trainData**[**0**].**size**();** **++**j**)**

**{**

cout **<<** trainData**[**i**][**j**]** **<<** "\t"**;**

**}**

cout **<<** endl**;**

**}**

**}**

//从文本文件"test.data"读取文本信息，然后创建训练用数据集

void DTA\_createTestDataset**()**

**{**

int i**,**j**;** //设置循环变量

int num**;**

char p**;**

string line**;**

string word**;**

vector**<**string**>** temp\_line**;**

ifstream fin**(**"test.data"**);**

getline**(**fin**,**line**);**

**while** **(**fin**.**get**(**p**))**

**{**

getline**(**fin**,**line**);**

**for** **(**string**::**size\_type i **=** 0**;**i **<** line**.**size**();** **++**i**)**

**{**

**if(**line**[**i**]==**' ' **||** line**[**i**]==**'\t'**)**

**{**

**if(!**word**.**empty**())**

**{**

temp\_line**.**push\_back**(**word**);**

word **=** ""**;**

**}**

**}**

**else**

word **+=** line**[**i**];**

**}**

temp\_line**.**push\_back**(**word**);**

word **=** ""**;**

testData**.**push\_back**(**temp\_line**);**

temp\_line**.**clear**();**

**}**

fin**.**close**();** //所以需要将最后一行的数据再填入到二维动态数组中

**for** **(**i **=** 0**;**i **<** testData**.**size**();** **++**i**)**

**{**

**for** **(**j **=** 0**;**j **<** testData**[**0**].**size**();** **++**j**)**

**{**

cout **<<** testData**[**i**][**j**]** **<<** "\t"**;**

**}**

cout **<<** endl**;**

**}**

**}**

//计算给定数据集的信息熵

double DTA\_calcEntropy **(**vector**<** vector**<**string**>** **>** **&**data**)**

**{**

int n **=** data**.**size**();** //设置循环范围

int i**;** //循环变量

map**<**string**,** int**>** classCounts**;** //设置键值对 key -- value 使用map

int label **=** data**[**0**].**size**()** **-** 1**;** //去除每一行最后一个的坐标

**for(**i **=** 0**;** i **<** n**;** i**++)** //初始为0

**{**

classCounts**[** data**[**i**][**label**]** **]** **=** 0**;**

**}**

**for(**i **=** 0**;** i **<** data**.**size**();** i**++)** //每当出现一次，+1

**{**

classCounts**[** data**[**i**][**label**]** **]** **+=** 1**;**

**}**

/\*计算信息熵

信息熵计算方法： Entropy(D) = -sum( p\_i \* log\_2(p\_i) )

D为样本集合

1<= i <= 样本D中所包含的类别数

p\_i 为第i个类别在D中所占的比例

\*/

double entropy **=** 0**;** //信息熵

map**<**string**,** int**>::**iterator it**;** //迭代器遍历map

**for(**it **=** classCounts**.**begin**();** it **!=** classCounts**.**end**();** it**++)**

**{**

double prob **=** **(**double**)(**it**->**second**)** **/** **(**double**)**n**;**

entropy **-=** prob **\*** **(** log**(**prob**)** **/** log**(**2**)** **);**

**}**

// cout << "shanno is " << entropy << endl;

// cout << "TA\_calcEntropyt is completed" << endl;

**return** entropy**;**

**}**

//按照给定特征划分数据集，新的数据集的维度少了一个 /axis ：特征下标 /value：特征值

vector**<** vector**<**string**>** **>** DTA\_splitDataSet**(**vector**<** vector**<**string**>** **>** data**,** int axis**,** string value**)**

**{**

vector**<** vector**<**string**>** **>** result**;** //新建vector储存result

**for(**int i **=** 0**;** i **<** data**.**size**();** i**++)**

**{**

**if(**data**[**i**][**axis**]** **==** value**)**

**{**

//将“当前特征”这个维度去掉

vector**<**string**>** removed**(**data**[**i**].**begin**(),** data**[**i**].**begin**()+**axis**);**

//用法：在指定位置it前“插入”区间[start, end)的所有元素.

removed**.**insert**(**removed**.**end**(),** data**[**i**].**begin**()+**axis**+**1**,** data**[**i**].**end**());**

result**.**push\_back**(**removed**);**

**}**

**}**

// cout << "DTA\_splitDataSet is completed" << endl;

**return** result**;**

**}**

//创建特征列表

vector**<**string**>** DTA\_createFeatureList**(**vector**<** vector**<**string**>** **>** **&**data**,** int axis**)**

**{**

int n **=** data**.**size**();**

vector**<**string**>** featureList**;** //特征的所有取值

set**<**string**>** s**;** //通过set进行去重操作

**for(**int j **=** 0**;** j **<** n**;** j**++)** //寻找该特征的所有可能取值

s**.**insert**(**data**[**j**][**axis**]);**

set**<**string**>::**iterator it**;** //之后两步将set集合压入featureList

**for(**it **=** s**.**begin**();** it **!=** s**.**end**();** it**++)**

**{**

featureList**.**push\_back**(\***it**);**

**}**

// cout << "DTA\_createFeatureList is completed" << endl;

**return** featureList**;**

**}**

//选择最好的数据集划分方式

int DTA\_chooseBestFeatureToSplit**(**vector**<** vector**<**string**>** **>** **&**data**)**

**{**

int n **=** data**[**0**].**size**()** **-** 1**;** //设置循环区间

double bestEntropy **=** DTA\_calcEntropy**(**data**);** //初始信息熵

double bestInfoGain **=** 0**;** //最大的信息增益

int bestFeature **=** 0**;** //最好的特征

**for(**int i **=** 0**;** i **<** n**;** i**++)** //所有特征

**{**

double newEntropy **=** 0**;** //新的特征信息熵

vector**<**string**>** featureList **=** DTA\_createFeatureList**(**data**,** i**);** //该特征的所有可能取值

**for(**int j **=** 0**;** j **<** featureList**.**size**();** j**++)**

**{**

//去除该特征

vector**<** vector**<**string**>** **>** subData **=** DTA\_splitDataSet**(**data**,** i**,** featureList**[**j**]);**

double prob **=** **(**double**)**subData**.**size**()** **/** **(**double**)**data**.**size**();**

newEntropy **+=** prob **\*** DTA\_calcEntropy**(**subData**);**

**}**

double infoGain **=** bestEntropy **-** newEntropy**;** //信息增益，即熵的减少，或数据无序度的减少

//选择最优的信息增益

**if(**infoGain **>** bestInfoGain**)**

**{**

bestInfoGain **=** infoGain**;**

bestFeature **=** i**;**

**}**

**}**

// cout << "DTA\_chooseBestFeatureToSplit is completed" << endl;

// cout << "the bestFeature is " << bestFeature << endl;

**return** bestFeature**;**

**}**

//返回出现次数最多的分类名称

//如果类标签依然不是唯一的，采用多数表决的方法定义叶子节点的分类

string DTA\_majorityCnt**(**vector**<**string**>** **&**classList**)**

**{**

int n **=** classList**.**size**();** //设置循环范围

map**<**string**,** int**>** classCount**;** //创建classCount的key -- value键值对，用来投票计数

int i**;** //循环变量i

//初始化

**for(**i **=** 0**;** i **<** n**;** i**++)**

**{**

classCount**[**classList**[**i**]]** **=** 0**;**

**}**

//投票计数

**for(**i **=** 0**;** i **<** n**;** i**++)**

**{**

classCount**[**classList**[**i**]]** **+=** 1**;**

**}**

//多数投票算法（Majority Vote Algorithm）

int DTA\_majorityCnt **=** 0**;** //最多票数计数器

map**<**string**,** int**>::**iterator it**;** //设置迭代器

string result **=** ""**;** //初始化result

**for(**it **=** classCount**.**begin**();** it **!=** classCount**.**end**();** it**++)**

**{**

**if(**it**->**second **>** DTA\_majorityCnt**)**

**{**

//it->first为键值；

//it->second为数值；

DTA\_majorityCnt **=** it**->**second**;**

result **=** it**->**first**;**

**}**

**}**

// cout << "DTA\_majorityCnt is complete" << endl;

// cout << "result is " << result << endl;

**return** result**;**

**}**

//递归构建决策树（核心算法）

Node**\*** DTA\_createTree**(**Node **\***root**,** vector**<** vector**<**string**>** **>** **&**data**,** vector**<**string**>** **&**attribute**)**

**{**

**if(**root **==** **NULL)** //新建立根结点root

root **=** **new** Node**();**

vector**<**string**>** classList**;** //特征列表

set**<**string**>** classList\_removal**;** //去重特征列表

int i**,** j**;** //循环变量

int label **=** data**[**0**].**size**()** **-** 1**;** //设置循环范围

int n **=** data**.**size**();** //数据集的大小

//TODO: move the n to the beginning as a global value(is ok)

**for(**i **=** 0**;** i **<** n**;** i**++)**

**{**

classList**.**push\_back**(**data**[**i**][**label**]);**

classList\_removal**.**insert**(**data**[**i**][**label**]);**

**}**

**if(**classList\_removal**.**size**()** **==** 1**)** //如果所有实例都属于同一类，停止划分

**{**

**if(**classList**[**0**]** **==** "yes"**)**

root**->**attribute **=** "yes"**;**

**else**

root**->**attribute **=** "no"**;**

root**->**isLeaf **=** **true;**

**return** root**;**

**}**

**if(**data**[**0**].**size**()** **==** 1**)** //遍历完所有特征，采用多数表决返回出现次数最多的类别

**{**

root**->**attribute **=** DTA\_majorityCnt**(**classList**);**

**return** root**;**

**}**

int bestFeatureIndex **=** DTA\_chooseBestFeatureToSplit**(**data**);** //选择最好的方式进行划分

vector**<**string**>** featureList **=** DTA\_createFeatureList**(**data**,** bestFeatureIndex**);** //得到该属性的所有可能值

string bestFeature **=** attribute**[**bestFeatureIndex**];** //记录最好的属性

root**->**attribute **=** bestFeature**;** //记录要划分的属性

**for(**i**=**0**;** i**<**featureList**.**size**();** i**++)** //对于当前属性的每个可能值，创建新的分支

**{**

vector**<**string**>** subAttribute**;** //新的属性列表，不包含当前要划分的属性，特征减少一个

**for(**j**=**0**;** j**<**attribute**.**size**();** j**++)**

**{**

**if(**bestFeature **!=** attribute**[**j**])**

subAttribute**.**push\_back**(**attribute**[**j**]);**

**}**

Node **\***newNode **=** **new** Node**();** //新建节点

newNode**->**val **=** featureList**[**i**];** //记录属性的取值

//递归创建决策树

update **=** DTA\_splitDataSet**(**data**,** bestFeatureIndex**,** featureList**[**i**]);**

DTA\_createTree**(**newNode**,** update**,** subAttribute**);**

root**->**childs**.**push\_back**(**newNode**);** //为父节点添加child

**}**

**return** root**;**

**}**

//打印决策树

void DTA\_printTree**(**Node **\***root**,** int depth**)**

**{**

ofstream fout**(**"DecisionTree.txt"**,**ios**::**app**);** //使用追加的方式打开文件

int i**;** //循环变量i

//输出算法

**for(**i **=** 0**;** i **<** depth**;** i**++)**

fout **<<** "\t"**;**

**if(**root**->**val **!=** ""**)**

**{**

fout **<<** root**->**val **<<** endl**;**

**for(**i **=** 0**;** i **<** depth**+**1**;** i**++)**

fout **<<** "\t"**;**

**}**

fout **<<** root**->**attribute **<<** endl**;**

fout**.**close**();**

//清空缓存区，防止未录入

//迭代器输出树

vector**<**Node**\*>::**iterator it**;**

**for(**it **=** root**->**childs**.**begin**();** it **!=** root**->**childs**.**end**();** it**++)**

**{**

DTA\_printTree**(\***it**,** depth**+**1**);**

**}**

// cout << "DTA\_printTree is completed" << endl;

**}**

string DTA\_classify**(**Node **\***root**,** vector**<**string**>** **&**attribute**,** vector**<**string**>** **&**test**)**

**{**

string firstFeature **=** root**->**attribute**;**

int firstFeatureIndex**;**

int i**;**

**for(**i**=**0**;** i**<**attributes**.**size**();** i**++)** //找到根节点是第几个特征

**{**

**if(**firstFeature **==** attribute**[**i**])**

**{**

firstFeatureIndex **=** i**;**

**break;**

**}**

**}**

**if(**root**->**isLeaf**)** //如果是叶子节点，直接输出结果

**return** root**->**attribute**;**

**for(**i**=**0**;** i**<**root**->**childs**.**size**();** i**++)**

**{**

**if(**test**[**firstFeatureIndex**]** **==** root**->**childs**[**i**]->**val**)**

**{**

**return** DTA\_classify**(**root**->**childs**[**i**],** attribute**,** test**);**

**}**

**}**

**}**

//将结果打印到“result.txt”中

void DTA\_printResult**(**vector**<** vector**<**string**>** **>** **&**data**,**Node**\*** root**)**

**{**

vector**<**string**>** temp**;**

string result**;**

int i **=** 1**;**

**try{**

vector**<**vector**<**string**>** **>** **::**iterator iter**;**

**for(**iter **=** data**.**begin**();** iter **!=** data**.**end**();** **++**iter**)**

**{**

ofstream fout**(**"result.txt"**,**ios**::**app**);**

temp **=** **\***iter**;**

result **=** DTA\_classify**(**root**,**attributes**,**temp**);**

**if(**result **!=** "no"**)**

result **=** "yes"**;**

fout **<<** "第" **<<** i **<<** "个测试的结果为：" **<<** result **<<** endl**;**

i**++;**

fout**.**close**();**

**}**

**}** **catch(**std**::**bad\_alloc**)**

**{**

cout **<<** "bad\_alloc happened" **<<** endl**;**

**}**

**}**

//释放节点

void DTA\_freeNode**(**Node **\***root**)**

**{**

**if(**root **==** **NULL)**

**return;**

vector**<**Node**\*>::**iterator it**;**

**for(**it**=**root**->**childs**.**begin**();** it **!=** root**->**childs**.**end**();** it**++)**

DTA\_freeNode**(\***it**);**

**delete** root**;**

**}**

//main函数

int main**(**void**)**

**{**

// 测试printResult的代码

// int i; //循环

// string result;

// vector <string> test;

// test.push\_back("雨");

// test.push\_back("冷");

// test.push\_back("正常");

// test.push\_back("无");

DTA\_createTrainDataset**();**

cout **<<** "DTA\_createTrainDataset is completed" **<<** endl**;**

DTA\_createTestDataset**();**

cout **<<** "DTA\_createTestDataset is completed" **<<** endl**;**

//获取root

root **=** DTA\_createTree**(**root**,** trainData**,** attributes**);**

cout **<<** "DTA\_createTree is completed" **<<** endl**;**

DTA\_printTree**(**root**,** 0**);**

cout **<<** "DTA\_printTree is completed" **<<** endl**;**

// result = DTA\_classify(root, attributes, test);

// cout << "测试的结果为：" << result << endl;

DTA\_printResult**(**testData**,**root**);**

cout **<<** "DTA\_printResult is completed" **<<** endl**;**

DTA\_freeNode**(**root**);**

cout **<<** "DTA\_freeNode is completed" **<<** endl**;**

**return** 0**;**

**}**