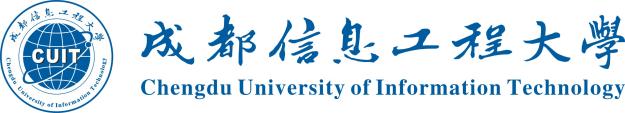
****

**实 验 报 告**

|  |  |
| --- | --- |
| **实验课程：** | **人工智能导论** |
| **实验项目：** | **实验三 使用决策树预测** |
| **指导教师：** |  |
| **学生姓名:** |  |
| **学生学号：** |  |
| **班 级：** |  |
| **分 数：** |  |

20 年月日

成都信息工程大学 计算机学院

## 问题描述

**使用决策树预测西瓜类型**

在给定的watermelon.txt数据集中，最后一列是西瓜是否好瓜的分类标签。参考题目1的示例程序完成西瓜的决策树分类算法。

## 解决方案阐述

选择一个变量，根据这个变量的不同取值，

把数据划分成不同子集，每个子集的熵都比原来的集合熵更低，

前后熵差越高，分类效果越好。

贪心选择当前最佳变量，构造成树结构，就实现了决策树

## 设计算法描述

当 当前节点不纯：

* 1. 计算当前节点的类别熵
  2. 计算当前节点的属性熵
  3. 计算各个属性的信息增益
  4. 计算各个属性的分类信息度量
  5. 计算各个属性的信息增益率

设当前节点为叶子节点

ID3算法的不足

ID3算法虽然提出了新思路，但是还是有很多值得改进的地方。

　　a) ID3没有考虑连续特征，比如长度，密度都是连续值，无法在ID3运用。这大大限制了ID3的用途。

　　b) ID3采用信息增益大的特征优先建立决策树的节点。很快就被人发现，在相同条件下，取值比较多的特征比取值少的特征信息增益大。比如一个变量有2个值，各为1/2，另一个变量为3个值，各为1/3，其实他们都是完全不确定的变量，但是取3个值的比取2个值的信息增益大。如果校正这个问题呢？

　　c) ID3算法对于缺失值的情况没有做考虑

　　d) 没有考虑过拟合的问题

## 算法实现（即完整源程序，带注解）

require.h

#ifndef REQUIRE\_H\_

#define REQUIRE\_H\_

#include "decisionstructure.h"

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <stack>

#include <string>

#include <map>

#include <vector>

#include <windows.h>

inline void assure(std::ifstream& in,const std::string& filename = "")

{

using namespace std;

if (!in)

{

fprintf(stderr, "Could not open file: %s\n", filename.c\_str());

exit(1);

}

}

inline void assure(std::ofstream& out,const std::string& filename = "")

{

using namespace std;

if (!out)

{

fprintf(stderr, "Could not open file: %s\n", filename.c\_str());

}

}

void load\_file(std::vector<Watermelon>& datas,std::vector<std::string>& attributes,const std::string& filename)

{

std::ifstream istrm(filename);

assure(istrm, filename);

if (!istrm.is\_open())

return;

char buffer[128];

const char \*attribute;

if(!istrm.eof())

{

istrm.getline(buffer, 128);

attribute = strtok(buffer, " ");

attributes.push\_back(std::string(attribute));

for(int i=0;i<7;++i)

{

attribute = strtok(NULL, " ");

attributes.push\_back(std::string(attribute));

}

}

while (!istrm.eof())

{

istrm.getline(buffer, 128);

Watermelon data;

attribute = strtok(buffer, " ");

\*(data.chooseMember(attributes[0]))=attribute;

for(int i=1;i<=7;++i)

{

attribute = strtok(NULL, " ");

\*(data.chooseMember(attributes[i]))=attribute;

}

datas.push\_back(data);

}

}

void match\_properties(std::vector<Watermelon> datas,std::vector<std::string> attributes,std::map<std::string, std::vector<std::string>>& map\_attr)

{

int index = 0;

for (auto attribute : attributes)

{

std::vector<std::string> attrTmp;

for (auto data : datas)

{

if (!attrTmp.empty() && std::find(attrTmp.begin(), attrTmp.end(), \*(data.chooseMember(attributes[index]))) == attrTmp.end())

attrTmp.push\_back(\*(data.chooseMember(attributes[index])));

else if (attrTmp.empty())

attrTmp.push\_back(\*(data.chooseMember(attributes[index])));

}

index++;

map\_attr[attribute] = attrTmp;

}

}

bool belongs\_same\_label(std::vector<Watermelon> datas,std::string label = "yes")

{

if (datas.empty())

{

std::cout << "the datas is empty" << std::endl;

exit(1);

}

for (auto iter = datas.begin(); iter != datas.end(); ++iter)

{

if (iter->good != label)

return false;

}

return true;

}

std::string majority\_of\_category(std::vector<Watermelon> datas)

{

int positivecategory = 0;

int negativecategory = 0;

for (auto data : datas)

{

if (data.good == "yes")

++positivecategory;

else if (data.good == "no")

++negativecategory;

}

return (positivecategory > negativecategory ? "yes" : "no");

}

//信息熵

double calculate\_information\_entropy(std::vector<Watermelon> datas,std::string mapAttr = "",std::string attribute = "good")

{

// Ent(D) = -∑(k=1, |Y|) p\_k \* log2(p\_k)

//

int size = 0;

int positive = 0;

int negative = 0;

// Beacuse of the datas only have two label.

// So entropy = positiveSample + negativeSample

//

if (attribute == "good")

{

size = datas.size();

auto judget = [&](Watermelon wm) { if (wm.good == "yes") { ++positive; } else { ++negative; }};

for\_each(datas.begin(), datas.end(), judget);

}

else

{

for (auto data : datas)

{

if (\*(data.chooseMember(attribute)) == mapAttr)

{

if (data.good == "yes")

++positive;

else

++negative;

++size;

}

}

}

if (positive == 0 || negative == 0)

return 0;

else

return -(((double)positive / size) \* log2((double)positive / size) + ((double)negative / size) \* log2((double)negative / size));

}

//比率

double proportion(std::vector<Watermelon>& datas,std::string mapAttr = "",std::string attribute = "")

{

int size = datas.size();

double proportion = 0;

for (auto data : datas)

{

if (\*(data.chooseMember(attribute)) == mapAttr)

++proportion;

}

proportion /= size;

return proportion;

}

//信息增益=训练集的信息熵-Σ划分信息熵

double calculate\_information\_gain(std::vector<Watermelon>& datas,std::string attribute,std::map<std::string, std::vector<std::string>> map\_attr)

{

// Gain(D, a) = Ent(D) - ∑(v=1, V) |D^|/|D| \* Ent(D)

//

double gain = calculate\_information\_entropy(datas);//训练集的信息熵

std::vector<std::string> attrs = map\_attr[attribute];

for (auto attr : attrs)

gain -= proportion(datas, attr, attribute) \* calculate\_information\_entropy(datas, attr, attribute);

//划分信息熵=比率\*按属性分的信息熵

return gain;

}

//信息增益率

double calculate\_information\_gain\_ratio(std::vector<Watermelon>& datas,std::string attribute,std::map<std::string, std::vector<std::string>> map\_attr)

{

// Gain\_ratio(D, a) = Gain(D, a) / IV(a)

//

double gain = calculate\_information\_gain(datas, attribute, map\_attr);

double iv = 0;

std::vector<std::string> attrs = map\_attr[attribute];

double tmp;

for (auto attr : attrs)

{

tmp=proportion(datas, attr, attribute);//分裂信息

iv -= tmp \* log2(tmp);

}

double gain\_ratio = gain / iv;

return gain\_ratio;

}

//最佳属性

std::pair<std::string, std::vector<std::string>> optimal\_attribute(std::vector<Watermelon>& datas,std::vector<std::string>& attributes,std::map<std::string, std::vector<std::string>> map\_attr)

{

std::map<std::string, double> map\_gains;

std::map<std::string, double> map\_gains\_ratio;

for (auto attribute : attributes)

{

map\_gains[attribute] = calculate\_information\_gain(datas, attribute, map\_attr);

map\_gains\_ratio[attribute] = calculate\_information\_gain\_ratio(datas, attribute, map\_attr);

}

// Sort the information gain and select the attribute of the maximum

// information gain.The biggest value is in the first.

//

std::vector<std::pair<std::string, double>> vec\_map\_gains(map\_gains.begin(), map\_gains.end());

std::vector<std::pair<std::string, double>> vec\_map\_gains\_ratios(map\_gains\_ratio.begin(), map\_gains\_ratio.end());

auto compare\_x\_y = [](const std::pair<std::string, double> x, const std::pair<std::string, double> y) {

return x.second > y.second;

};

std::sort(vec\_map\_gains.begin(), vec\_map\_gains.end(), compare\_x\_y);

std::sort(vec\_map\_gains\_ratios.begin(), vec\_map\_gains\_ratios.end(), compare\_x\_y);

// Find information gain greater than average.

//

std::vector<std::string> vec\_map\_gains\_name;

int vec\_map\_gains\_size = vec\_map\_gains.size() / 2;

for (int i = 0; i < vec\_map\_gains\_size; ++i)

vec\_map\_gains\_name.push\_back(vec\_map\_gains[i].first);

std::string best\_attribute;

for (auto vec\_map\_gains\_ratio : vec\_map\_gains\_ratios)

{

if (std::find(vec\_map\_gains\_name.begin(), vec\_map\_gains\_name.end(), vec\_map\_gains\_ratio.first)!= vec\_map\_gains\_name.end())

{

best\_attribute = vec\_map\_gains\_ratio.first;

break;

}

}

if (!best\_attribute.empty())

{

auto search = map\_attr.find(best\_attribute);

if (search != map\_attr.end())

return std::make\_pair(search->first, search->second);

else

return std::make\_pair(std::string(""), std::vector<std::string>());

}

else

return std::make\_pair(std::string(""), std::vector<std::string>());

}

std::vector<Watermelon> remain\_watermelon\_datas(std::vector<Watermelon> datas,std::string mapAttr,std::string attribute)

{

std::vector<Watermelon> tmp;

for (auto data : datas)

{

if (\*(data.chooseMember(attribute)) == mapAttr)

tmp.push\_back(data);

}

return tmp;

}

void print\_tree(TreeRoot pTree, int depth)

{

for (int i = 0; i < depth; ++i) {

std::cout << '\t';

}

if (!pTree->edgeValue.empty()) {

std::cout << "--" << pTree->edgeValue << "--" << std::endl;

for (int i = 0 ; i < depth; ++i) {

std::cout << '\t';

}

}

std::cout << pTree->attribute << std::endl;

for (auto child : pTree->childs) {

print\_tree(child, depth + 1);

}

}

void Test(TreeRoot p,Watermelon data)

{

if(p->attribute=="yes" || p->attribute=="no")

{

std::cout<<p->attribute<<std::endl;

return;

}

for(auto child:p->childs)

{

if(child->edgeValue.empty())

continue;

if(\*(data.chooseMember(p->attribute))==child->edgeValue)

{

Test(child,data);

return;

}

}

std::cout<<"no"<<std::endl;

}

void test\_tree(TreeRoot pTree,std::vector<std::string> attributes)

{

std::cout<<"please input a watermelon's information,including";

for(auto attri:attributes)

std::cout<<" "<<attri;

std::cout<<".if some attribute is unknown,please input #."<<std::endl;

Watermelon data;

std::string attri;

while(1)

{

std::cout<<"if you want quit test,please input 1"<<std::endl;

int op;

std::cin>>op;

if(op==1)

break;

for(int i=0;i<6;++i)

{

std::cin>>attri;

\*(data.chooseMember(attributes[i]))=attri;

}

Test(pTree,data);

system("pause");

}

}

#endif

Decisionstructure.h

#ifndef DECISIONSTRUCTURE\_H\_

#define DECISIONSTRUCTURE\_H\_

#include <iostream>

#include <string>

#include <vector>

// 西瓜个体定义

class Watermelon

{

public:

std::string id; // 编号

std::string color; // 颜色

std::string pedicle; // 根蒂

std::string sound; // 声音

std::string texture; // 纹理

std::string umbilical; // 脐部

std::string touch; // 触感

std::string good; // 好瓜

friend

std::ostream& operator<<(std::ostream& os, const Watermelon& wm)

{

os << wm.id << " " << wm.color << " " << wm.pedicle << " "

<< wm.sound << " " << wm.texture << " " << wm.umbilical << " "

<< wm.touch << " " << wm.good;

os << std::endl;

return os;

}

std::string\* chooseMember(const std::string &s)

{

if(s=="id")

return &this->id;

else if(s=="color")

return &this->color;

else if(s=="pedicle")

return &this->pedicle;

else if(s=="sound")

return &this->sound;

else if(s=="texture")

return &this->texture;

else if(s=="umbilical")

return &this->umbilical;

else if(s=="touch")

return &this->touch;

else if(s=="good")

return &this->good;

else

return nullptr;

}

};

struct Node

{

std::string attribute;

std::string edgeValue;

std::vector<Node\*> childs;

Node() : attribute(""), edgeValue("") { }

};

template<typename T>

using Ptr = T\*;

using TreeRoot = Ptr<Node>;

#endif

C4.5.cpp

#include "require.h"

#include "decisionstructure.h"

#include <iostream>

#include <string>

#include <vector>

// C4.5决策算法 -- 西瓜决策树

TreeRoot TreeGenerate(TreeRoot pTree,

std::vector<Watermelon> datas, // 训练集

std::vector<std::string> attributes, // 属性集

std::map<std::string, std::vector<std::string>> map\_attr)

{

if (belongs\_same\_label(datas, "yes"))

{

// All samples are positive.

pTree->attribute = "yes";

return pTree;

}

else if (belongs\_same\_label(datas, "no"))

{

// All samples are negative.

pTree->attribute = "no";

return pTree;

}

std::pair<std::string, std::vector<std::string>> optimal\_attrs = optimal\_attribute(datas, attributes, map\_attr);

pTree->attribute = optimal\_attrs.first;

for (auto aptimal\_attr : optimal\_attrs.second)

{

Node\* new\_node = new Node();

new\_node->edgeValue = aptimal\_attr;

std::vector<Watermelon> new\_datas = remain\_watermelon\_datas(datas, aptimal\_attr, optimal\_attrs.first);

if (new\_datas.empty())

new\_node->attribute = majority\_of\_category(datas);

else

{

std::vector<std::string> new\_attributes;

for (auto train\_attribute : attributes)

{

if(train\_attribute.compare(optimal\_attrs.first))

new\_attributes.push\_back(train\_attribute);

}

TreeGenerate(new\_node, new\_datas, new\_attributes, map\_attr);

}

pTree->childs.push\_back(new\_node);

}

return pTree;

}

int main()

{

std::vector<Watermelon> datas;

std::vector<std::string> attributes;

std::vector<std::string> train\_attributes;

train\_attributes.push\_back("color");

train\_attributes.push\_back("pedicle");

train\_attributes.push\_back("sound");

train\_attributes.push\_back("texture");

train\_attributes.push\_back("umbilical");

train\_attributes.push\_back("touch");

// load the datas and attributes..

//

load\_file(datas, attributes, "watermelondatas.txt");

std::map<std::string, std::vector<std::string>> map\_attr;

// match attribute set..

//

match\_properties(datas, attributes, map\_attr);

TreeRoot pTree = new Node();

pTree = TreeGenerate(pTree, datas, train\_attributes, map\_attr);

print\_tree(pTree, 0);

test\_tree(pTree,train\_attributes);

return 0;

}

/\*

jetblack slightlycurled turbid lightblur slightlysunken softsticky

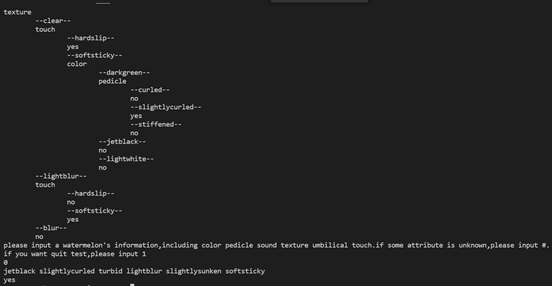
yes

jetblack slightlycurled turbid clear slightlysunken softsticky

no

\*/

## 实验结果测试与分析



## 思考及学习心得

在本次实验中，我对决策树有了更深一步的认识，同时也了解了ID3。巩固了理论知识，能给完成其代码实现，计算出特征数据的熵、信息增益以及信息增益率。并且增加了我对cpp的理解。

通过上述的实验结果可以看出，所写代码能够完成题目的要求计算内容。通过完成实验，我了解到了各个值的作用。信息熵可以理解为信息的不确定性，信息熵越大，表明越不确定的东西。信息熵衡量的是一个结点当中的纯度，而我们要的是一个衡量纯度的属性，这里就要用到信息增益，即信息增益是衡量一个属性把当前样本集进行划分之后，这个纯度的变化量，所以要信息增益越大越好。而当我们查看信息增益的公式，发现属性的可能取值种类有多种，而种类数量影响决策树的性能。具体来说就是，信息增益准则对可取数值较多的属性有所偏好。