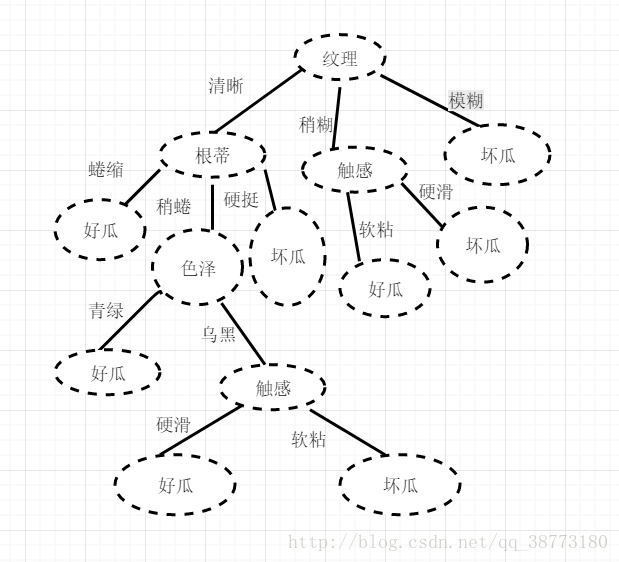
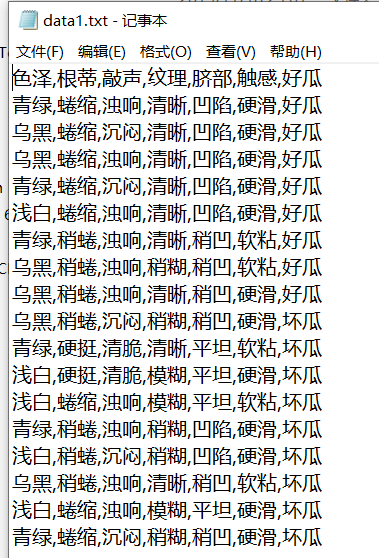
决策树的实现 java



测试数据：



引入信息熵作为度量样本集合不确定度的指标，采用信息增益这个量作为纯度的度量，选取信息增益最大的特征进行分裂

信息增量=信息熵-条件熵

实现代码

package xx;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

class treeNode{//树节点

private String sname;//节点名

public treeNode(String str) {

sname=str;

}

public String getsname() {

return sname;

}

ArrayList<String> label=new ArrayList<String>();//和子节点间的边标签

ArrayList<treeNode> node=new ArrayList<treeNode>();//对应子节点

}

public class ID3 {

private ArrayList<String> label=new ArrayList<String>();//特征标签

private ArrayList<ArrayList<String>> date=new ArrayList<ArrayList<String>>();//数据集

private ArrayList<ArrayList<String>> test=new ArrayList<ArrayList<String>>();//测试数据集

private ArrayList<String> sum=new ArrayList<String>();//分类种类数

private String kind;

public ID3(String path,String path0) throws FileNotFoundException {

//初始化训练数据并得到分类种数

getDate(path);

//获取测试数据集

gettestDate(path0);

init(date);

}

public void init(ArrayList<ArrayList<String>> date) {

//得到种类数

sum.add(date.get(0).get(date.get(0).size()-1));

for(int i=0;i<date.size();i++) {

if(sum.contains(date.get(i).get(date.get(0).size()-1))==false) {

sum.add(date.get(i).get(date.get(0).size()-1));

}

}

}

//获取测试数据集

public void gettestDate(String path) throws FileNotFoundException {

String str;

int i=0;

try {

//BufferedReader in=new BufferedReader(new FileReader(path));

FileInputStream fis = new FileInputStream(path);

InputStreamReader isr = new InputStreamReader(fis, "UTF-8");

BufferedReader in = new BufferedReader(isr);

while((str=in.readLine())!=null) {

String[] strs=str.split(",");

ArrayList<String> line =new ArrayList<String>();

for(int j=0;j<strs.length;j++) {

line.add(strs[j]);

//System.out.print(strs[j]+" ");

}

test.add(line);

//System.out.println();

i++;

}

in.close();

}catch(Exception e) {

e.printStackTrace();

}

}

//获取训练数据集

public void getDate(String path) throws FileNotFoundException {

String str;

int i=0;

try {

//BufferedReader in=new BufferedReader(new FileReader(path));

FileInputStream fis = new FileInputStream(path);

InputStreamReader isr = new InputStreamReader(fis, "UTF-8");

BufferedReader in = new BufferedReader(isr);

while((str=in.readLine())!=null) {

if(i==0) {

String[] strs=str.split(",");

for(int j=0;j<strs.length;j++) {

label.add(strs[j]);

//System.out.print(strs[j]+" ");

}

i++;

//System.out.println();

continue;

}

String[] strs=str.split(",");

ArrayList<String> line =new ArrayList<String>();

for(int j=0;j<strs.length;j++) {

line.add(strs[j]);

//System.out.print(strs[j]+" ");

}

date.add(line);

//System.out.println();

i++;

}

in.close();

}catch(Exception e) {

e.printStackTrace();

}

}

public double Ent(ArrayList<ArrayList<String>> dat) {

//计算总的信息熵

int all=0;

double amount=0.0;

for(int i=0;i<sum.size();i++) {

for(int j=0;j<dat.size();j++) {

if(sum.get(i).equals(dat.get(j).get(dat.get(0).size()-1))) {

all++;

}

}

if((double)all/dat.size()==0.0) {

continue;

}

amount+=((double)all/dat.size())\*(Math.log(((double)all/dat.size()))/Math.log(2.0));

all=0;

}

if(amount==0.0) {

return 0.0;

}

return -amount;//计算信息熵

}

//计算条件熵并返回信息增益值

public double condtion(int a,ArrayList<ArrayList<String>> dat) {

ArrayList<String> all=new ArrayList<String>();

double c=0.0;

all.add(dat.get(0).get(a));

//得到属性种类

for(int i=0;i<dat.size();i++) {

if(all.contains(dat.get(i).get(a))==false) {

all.add(dat.get(i).get(a));

}

}

ArrayList<ArrayList<String>> plus=new ArrayList<ArrayList<String>>();

//部分分组

ArrayList<ArrayList<ArrayList<String>>> count=new ArrayList<ArrayList<ArrayList<String>>>();

//分组总和

for(int i=0;i<all.size();i++) {

for(int j=0;j<dat.size();j++) {

if(true==all.get(i).equals(dat.get(j).get(a))) {

plus.add(dat.get(j));

}

}

count.add(plus);

c+=((double)count.get(i).size()/dat.size())\*Ent(count.get(i));

plus.removeAll(plus);

}

return (Ent(dat)-c);

//返回条件熵

}

//计算信息增益最大属性

public int Gain(ArrayList<ArrayList<String>> dat) {

ArrayList<Double> num=new ArrayList<Double>();

//保存各信息增益值

for(int i=0;i<dat.get(0).size()-1;i++) {

num.add(condtion(i,dat));

}

int index=0;

double max=num.get(0);

for(int i=1;i<num.size();i++) {

if(max<num.get(i)) {

max=num.get(i);

index=i;

}

}

//System.out.println("<"+label.get(index)+">");

return index;

}

//构建决策树

public treeNode creattree(ArrayList<ArrayList<String>> dat) {

int index=Gain(dat);

treeNode node=new treeNode(label.get(index));

ArrayList<String> s=new ArrayList<String>();//属性种类

s.add(dat.get(0).get(index));

//System.out.println(dat.get(0).get(index));

for(int i=1;i<dat.size();i++) {

if(s.contains(dat.get(i).get(index))==false) {

s.add(dat.get(i).get(index));

//System.out.println(dat.get(i).get(index));

}

}

ArrayList<ArrayList<String>> plus=new ArrayList<ArrayList<String>>();

//部分分组

ArrayList<ArrayList<ArrayList<String>>> count=new ArrayList<ArrayList<ArrayList<String>>>();

//分组总和

//得到节点下的边标签并分组

for(int i=0;i<s.size();i++) {

node.label.add(s.get(i));//添加边标签

//System.out.print("添加边标签:"+s.get(i)+" ");

for(int j=0;j<dat.size();j++) {

if(true==s.get(i).equals(dat.get(j).get(index))) {

plus.add(dat.get(j));

}

}

count.add(plus);

//System.out.println();

//以下添加结点

int k;

String str=count.get(i).get(0).get(count.get(i).get(0).size()-1);

for(k=1;k<count.get(i).size();k++) {

if(false==str.equals(count.get(i).get(k).get(count.get(i).get(k).size()-1))) {

break;

}

}

if(k==count.get(i).size()) {

treeNode dd=new treeNode(str);

node.node.add(dd);

//System.out.println("这是末端:"+str);

}

else {

//System.out.print("寻找新节点:");

node.node.add(creattree(count.get(i)));

}

plus.removeAll(plus);

}

return node;

}

//输出决策树

public void print(ArrayList<ArrayList<String>> dat) {

System.out.println("构建的决策树如下：");

treeNode node=null;

node=creattree(dat);//类

put(node);//递归调用

}

//用于递归的函数

public void put(treeNode node) {

System.out.println("结点："+node.getsname()+"\n");

for(int i=0;i<node.label.size();i++) {

System.out.println(node.getsname()+"的标签属性:"+node.label.get(i));

if(node.node.get(i).node.isEmpty()==true) {

System.out.println("叶子结点："+node.node.get(i).getsname());

}

else {

put(node.node.get(i));

}

}

}

//用于对待决策数据进行预测并将结果保存在指定路径

public void testdate(ArrayList<ArrayList<String>> test,String path) throws IOException {

treeNode node=null;

int count=0;

node=creattree(this.date);//类

try {

BufferedWriter out=new BufferedWriter(new FileWriter(path));

for(int i=0;i<test.size();i++) {

testput(node,test.get(i));//递归调用

//System.out.println(kind);

for(int j=0;j<test.get(i).size();j++) {

out.write(test.get(i).get(j)+",");

}

if(kind.equals(date.get(i).get(date.get(i).size()-1))==true) {

count++;

}

out.write(kind);

out.newLine();

}

System.out.println("该次分类结果正确率为："+(double)count/test.size()\*100+"%");

out.flush();

out.close();

}catch(IOException e) {

e.printStackTrace();

}

}

//用于测试的递归调用

public void testput(treeNode node,ArrayList<String> t) {

int index=0;

for(int i=0;i<this.label.size();i++) {

if(this.label.get(i).equals(node.getsname())==true) {

index=i;

break;

}

}

for(int i=0;i<node.label.size();i++) {

if(t.get(index).equals(node.label.get(i))==false) {

continue;

}

if(node.node.get(i).node.isEmpty()==true) {

//System.out.println("分类结果为："+node.node.get(i).getsname());

this.kind=node.node.get(i).getsname();//取出分类结果

}

else {

testput(node.node.get(i),t);

}

}

}

public static void main(String[] args) throws IOException {

String data="F:\\data1.txt";//训练数据集

String test="F:\\test.txt";//测试数据集

String result="F:\\result.txt";//预测结果集

ID3 id=new ID3(data,test);//初始化数据

id.print(id.date);//构建并输出决策树

//id.testdate(id.test,result);//预测数据并输出结果

}

}

运行结果：

