

#include <iostream>

#include <vector>

#include <string>

#include <map>

#include <set>

#include <algorithm>

#include <cmath>

using namespace std;

// 定义样本数据结构

struct Sample {

bool hasOption; // 是否有其他选择

bool isHungry; // 是否饥饿

string price; // 价格

string type; // 餐饮类型

string patron; // 餐厅顾客情况

string wait; // 等待时间

bool decision; // 决定（是/否）

};

// 定义ID3决策树的节点

struct Node {

string attribute;

map<string, Node\*> children;

bool isLeaf;

bool decision;

Node() : isLeaf(false) {}

};

// 准备样本数据集

vector<Sample> samples = {

{true, true, "SSS", "法式", "有人", "0-10", true},

{true, true, "S", "中餐", "客满", "30-60", false},

{false, true, "S", "快餐", "有人", "0-10", true},

{true, false, "S", "中餐", "客满", "10-30", true},

{true, false, "SSS", "法式", "客满", "0-10", false},

{false, true, "S", "意大利式", "有人", "0-10", true},

{false, false, "S", "快餐", "无人", "0-10", false},

{false, true, "$$", "中餐", "有人", "0-10", true},

{false, false, "$$$", "法式", "客满", ">60", false},

{true, true, "$$", "意大利式", "客满", "10-30", true},

{false, false, "$", "中餐", "无人", "0-10", false},

{false, true, "S", "快餐", "客满", "30-60", true}

};

// 计算信息熵

double entropy(const vector<Sample>& samples) {

map<bool, int> counts;

for (const auto& sample : samples) {

counts[sample.decision]++;

}

double total = samples.size();

double result = 0.0;

for (const auto& pair : counts) {

double p = pair.second / total;

result -= p \* log2(p);

}

return result;

}

// 计算信息增益

double information\_gain(const vector<Sample>& samples, const string& attr) {

map<string, vector<Sample>> subsets;

for (const auto& sample : samples) {

if (attr == "hasOption") {

subsets[sample.hasOption ? "是" : "否"].push\_back(sample);

}

else if (attr == "isHungry") {

subsets[sample.isHungry ? "是" : "否"].push\_back(sample);

}

else if (attr == "price") {

subsets[sample.price].push\_back(sample);

}

else if (attr == "type") {

subsets[sample.type].push\_back(sample);

}

else if (attr == "patron") {

subsets[sample.patron].push\_back(sample);

}

else if (attr == "wait") {

subsets[sample.wait].push\_back(sample);

}

}

double total\_entropy = entropy(samples);

double weighted\_entropy\_sum = 0.0;

for (const auto& pair : subsets) {

double subset\_entropy = entropy(pair.second);

weighted\_entropy\_sum += (pair.second.size() / (double)samples.size()) \* subset\_entropy;

}

return total\_entropy - weighted\_entropy\_sum;

}

// ID3算法实现

Node\* ID3(vector<Sample> samples, set<string> attributes) {

if (samples.empty()) return nullptr;

if (all\_of(samples.begin(), samples.end(), [](const Sample& s) { return s.decision; })) {

Node\* leaf = new Node();

leaf->isLeaf = true;

leaf->decision = true;

return leaf;

}

if (all\_of(samples.begin(), samples.end(), [](const Sample& s) { return !s.decision; })) {

Node\* leaf = new Node();

leaf->isLeaf = true;

leaf->decision = false;

return leaf;

}

if (attributes.empty()) {

Node\* leaf = new Node();

leaf->isLeaf = true;

leaf->decision = count\_if(samples.begin(), samples.end(), [](const Sample& s) { return s.decision; }) > samples.size() / 2;

return leaf;

}

string best\_attr;

double best\_gain = -1.0;

for (const auto& attr : attributes) {

double gain = information\_gain(samples, attr);

if (gain > best\_gain) {

best\_gain = gain;

best\_attr = attr;

}

}

Node\* node = new Node();

node->attribute = best\_attr;

map<string, vector<Sample>> subsets;

for (const auto& sample : samples) {

if (best\_attr == "hasOption") {

subsets[sample.hasOption ? "是" : "否"].push\_back(sample);

}

else if (best\_attr == "isHungry") {

subsets[sample.isHungry ? "是" : "否"].push\_back(sample);

}

else if (best\_attr == "price") {

subsets[sample.price].push\_back(sample);

}

else if (best\_attr == "type") {

subsets[sample.type].push\_back(sample);

}

else if (best\_attr == "patron") {

subsets[sample.patron].push\_back(sample);

}

else if (best\_attr == "wait") {

subsets[sample.wait].push\_back(sample);

}

}

attributes.erase(best\_attr);

for (const auto& pair : subsets) {

Node\* child = ID3(pair.second, attributes);

node->children[pair.first] = child;

}

return node;

}

// 分类函数

bool classify(Node\* root, const Sample& sample) {

if (root->isLeaf) {

return root->decision;

}

string attr = root->attribute;

if (attr == "hasOption") {

return classify(root->children[sample.hasOption ? "是" : "否"], sample);

}

else if (attr == "isHungry") {

return classify(root->children[sample.isHungry ? "是" : "否"], sample);

}

else if (attr == "price") {

return classify(root->children[sample.price], sample);

}

else if (attr == "type") {

return classify(root->children[sample.type], sample);

}

else if (attr == "patron") {

return classify(root->children[sample.patron], sample);

}

else if (attr == "wait") {

return classify(root->children[sample.wait], sample);

}

return false;

}

int main() {

set<string> attributes = { "hasOption", "isHungry", "price", "type", "patron", "wait" };

Node\* root = ID3(samples, attributes);

Sample test\_sample = { true, true, "SSS", "法式", "有人", "0-10", false };

bool result = classify(root, test\_sample);

cout << "分类结果: " << (result ? "是" : "否") << endl;

return 0;

}