厦門大學

软 件 学 院

《人工智能导论》实验报告

题　　目　决策树算法

姓　　名　陈澄

学　　号　32420212202930

班　　级　软工三班

实验时间　2024/04/16

2024 年 **04** 月 **16** 日

1 实验目的

编程实现决策树算法 ID3；理解算法原理。

2 实验内容

利用 taindata.txt 的数据（75*5，第 5 列为标签）进行训练，构造决策树；利用构造好的决策树对 testdata.txt 的数据进行分类，并输出分类准确率。

3 实验步骤

1.定义鸢尾花数据集数据结构 DataSet

```
class DataSet {
public:
    vector<string> Attribute; //属性标签: "花萼长度","花萼宽度","花萼长度","花瓣长度"
    vector<vector<string>> Data; // 属性值: 5.1 3.5 1.4 0.2
    map<string, vector<string>> AttributeValue; //映射类型, 整个表的数据: {"类别":{ ... },"花萼长度":{5.1,4.9,5.7....},"..":{ ... }}

    DataSet(); //初始构造函数
    DataSet(vector<vector<string>> data, vector<string> attribute); //过程的构造函数
    void Connect(); //关联属性标签与属性值
};
DataSet::DataSet() {
    Attribute = { "花萼长度","花萼宽度","花瓣长度","花瓣宽度" };
}
DataSet::DataSet(vector<vector<string>> data, vector<string> attribute) {
    Data = data;
    Attribute = attribute;
    Connect();
}
```

2.Connect()方法用于关联属性标签与属性值

```
void DataSet::Connect() {
    if (Data.empty()) return;

    vector<vector<string>> attribute; //存储属性值的转置
    vector<string> TempAttribute = Attribute; //存储属性标签
    TempAttribute.push_back("类别"); //5
    attribute.resize(TempAttribute.size()); //5

    // Data 150行x5列
    for (int i = 0; i < Data[0].size(); i++) {  //i 0-4
        for (int j = 0; j < Data.size(); j++) { //j 0-149
            attribute[i].push_back(Data[j][i]);
        }

        AttributeValue[TempAttribute[i]] = attribute[i];
    }

}
```

3.Read_file()方法用于读取数据集文件

```cpp
void Read_file(DataSet& dataSet, string fname) {
    // 读取文件
    ifstream file_data(fname, ios::in);
    if (!file_data.is_open()) {
        cout << "Error: opening file fail" << endl;
        exit(1);
    }
    else {
        string line;
        vector<string> words;
        string word;

        // 读取数据
        istringstream sin;
        // 按行读取数据
        while (getline(file_data, line))
        {
            word.clear();
            words.clear();
            sin.clear();
            sin.str(line);
            while (getline(sin, word, ' ')) {
                words.push_back(word);   //将每一格中的数据逐个push
            }
            dataSet.Data.push_back(words);
        }
        file_data.close();
    }
    dataSet.Connect();
}
```

4.定义决策树结点数据结构

```cpp
class Node {
public:
    Node() {
        isLeaf = false;
        isRoot = false;
        nodeAccuracy = 0;
    }

    vector<double> MidValue;
    bool isLeaf;
    bool isRoot;
    string node_Attribute;
    //判断标准
    double Mid;
    string Attribute;
    vector<Node*> ChildrenNode;
    double nodeAccuracy;

};
```

5.定义决策树类及其相关方法

```
class DecisionTree {
public:
    void TreeGenerate(DataSet& dataSet, Node* Father); //生成决策树，返回根结点的指针
    double CalcEntropy(DataSet& dataSet); //计算一个数据集的信息熵
    double CalcInfoGain(double midValue, DataSet& dataSet, string Value); //计算一个属性的信息增益

    double AccuracyRate(DataSet& dataSet, Node* node);
    vector<double> FindMidValue(DataSet& dataSet); //计算一个属性的划分点，构建循环计算一个属性的信息增益
    map<string, double> FindBestInfoGain(DataSet& dataSet); //找最大信息增益，返回最优属性以及最大增益的映射
    map<string, int> CountTimes(DataSet& dataSet); //计算各类别的样本数量
    void DestoryDecisionTree(Node* node);//删除决策树
    vector<string> Prediction(DataSet& dataSet, Node* node);
};
```

6.TreeGenerate()方法用于生成决策树

根据结点的不同情况选择是否分裂

情况 1

```
void DecisionTree::TreeGenerate(DataSet& dataSet, Node* Father) {

    Node* newNode = new Node;
    Father->ChildrenNode.push_back(newNode);
    vector<double> newMid = FindMidValue(dataSet);//本次的各属性最优划分点
    newNode->MidValue = newMid;

    // 情况1：如果数据集中数据属于同一类别，将node标记为C类叶结点
    bool isSame = true;
    string curAttr = dataSet.AttributeValue["类别"][0];
    for (int i = 0; i < dataSet.AttributeValue["类别"].size(); i++) {
        if (dataSet.AttributeValue["类别"][i] != curAttr) {
            isSame = false;
            break;
        }
    }
    if (isSame) {
        newNode->isLeaf = true;
        newNode->node_Attribute = curAttr;
        return;
    }
```

情况 2

```cpp
// 情况2: 属性集为空或数据集中样本在属性集上取值相同
isSame = true;
for (int i = 0; i < dataSet.Attribute.size(); i++) {
    string a = dataSet.AttributeValue[dataSet.Attribute[i]][0];
    if (stod(a) > Father->MidValue[i]) {//?
        for (int j = 1; j < dataSet.Data.size(); j++) {
            if (stod(dataSet.AttributeValue[dataSet.Attribute[i]][j]) <= Father->MidValue[i]) {
                isSame = false;
                break;
            }
        }
    }
    else {
        for (int j = 1; j < dataSet.Data.size(); j++) {
            if (stod(dataSet.AttributeValue[dataSet.Attribute[i]][j]) > Father->MidValue[i]) {
                isSame = false;
                break;
            }
        }
    }
    if (isSame == false) break;
}
if (dataSet.Attribute.empty() || isSame == true) {
    newNode->isLeaf = true;
    map<string, int> mp = CountTimes(dataSet);
    int maxTimes = -1;
    string maxValue;
    for (auto i = mp.begin(); i != mp.end(); i++) {
        if (i->second > maxTimes) {
            maxValue = i->first;
            maxTimes = i->second;
        }
    }
    newNode->node_Attribute = maxValue;
    return;
}
```

情况 3

```cpp
//情况3:从属性集中划分最优属性
map<string, double> BestAttributeMap = FindBestInfoGain(dataSet);
string BestAttribute = BestAttributeMap.begin()->first;
newNode->Attribute = BestAttribute;

//找出一个属性中的最优划分点
vector<string> TempAttribute;
TempAttribute.push_back(BestAttribute);
vector<string> TempValue = dataSet.AttributeValue[BestAttribute];
vector<string> TempAttr = dataSet.AttributeValue["类别"];
vector<vector<string>> TempData(TempValue.size());
for (int i = 0; i < TempValue.size(); i++) {
    TempData[i].push_back(TempValue[i]);
    TempData[i].push_back(TempAttr[i]);
}
DataSet d(TempData, TempAttribute);
vector<double> MidArray = FindMidValue(d);

double bestmid = MidArray[0];//最优属性的最优划分点
newNode->Mid = bestmid;
vector<vector<vector<string>>> NextData(2);//下一次分类的数据
for (int i = 0; i < dataSet.Data.size(); i++) {
    if (stod(dataSet.AttributeValue[BestAttribute][i]) < bestmid) {
        NextData[0].push_back(dataSet.Data[i]);
    }
    else {
        NextData[1].push_back(dataSet.Data[i]);
    }
}
```

情况 4

```cpp
for (int i = 0; i < NextData.size(); i++) {
    Node* newChild = new Node;

    if (NextData[i].empty()) {
        newNode->ChildrenNode.push_back(newChild);
        newChild->isLeaf = true;
        map<string, int>mp = CountTimes(dataSet);
        int maxTimes = -1;
        string maxValue;
        for (auto i = mp.begin(); i != mp.end(); i++) {
            if (i->second > maxTimes) {
                maxValue = i->first;
                maxTimes = i->second;
            }
        }
        newChild->node_Attribute = maxValue;

        return;
    }
    else {
        //情况4: 需要预剪枝
        DataSet NewDataSet(NextData[i], dataSet.Attribute);

        //对newNode的属性初始化
        map<string, int> mp = CountTimes(dataSet);
        int maxTimes = -1;
        string maxValue;
        for (auto i = mp.begin(); i != mp.end(); i++) {
            if (i->second > maxTimes) {
                maxValue = i->first;
                maxTimes = i->second;
            }
        }
        newNode->node_Attribute = maxValue;

        //对newChild初始化
        mp = CountTimes(NewDataSet); //先标记为该数据集中数量最多的类
        maxTimes = -1;
        maxValue = "";
        for (auto i = mp.begin(); i != mp.end(); i++) {
            if (i->second > maxTimes) {
                maxValue = i->first;
                maxTimes = i->second;
            }
        }
        newChild->node_Attribute = maxValue;

        if (AccuracyRate(dataSet, newNode) > AccuracyRate(NewDataSet, newChild)
            newNode->isLeaf = true;
            delete newChild;
            return;
        }
        delete newChild;
        TreeGenerate(NewDataSet, newNode);
    }
}
```

7.CalcEntropy()信息熵计算方法

```cpp
double DecisionTree::CalcEntropy(DataSet& dataSet) {
    int sum = dataSet.Data.size();//总数据数
    map<string, int> ClassCount = CountTimes(dataSet);

    vector<string> classList = dataSet.AttributeValue["类别"];
    double entropy = 0;//信息熵

    for (auto item = ClassCount.begin(); item != ClassCount.end(); item++) {
        double p = (double)item->second / sum;
        if (p == 0) entropy -= 0;
        else entropy -= p * log(p) / log(2);
    }
    return entropy;
}
```

8.FindMidValue()寻找连续数据最优划分点方法

```cpp
vector<double> DecisionTree::FindMidValue(DataSet& dataSet) {//找出属性中最优划分点
    vector<double> bestMid;

    for (int i = 0; i < dataSet.Attribute.size(); i++) {
        vector<string> valueString = dataSet.AttributeValue[dataSet.Attribute[i]];

        //将属性值从string类型转为double类型
        vector<double> valueDouble;
        for (int j = 0; j < valueString.size(); j++) {
            valueDouble.push_back(stod(valueString[j]));
        }

        //排序double类型
        sort(valueDouble.begin(), valueDouble.end());

        vector<double> midArray;
        for (int j = 0; j < valueDouble.size() - 1; j++) {
            midArray.push_back((valueDouble[j] + valueDouble[j + 1]) / 2);
        }

        double bestmid;
        if (midArray.empty()) bestmid = valueDouble[0];
        else bestmid = midArray[0]; //初始化中间点
        double maxmidEntrophy = 0; //划分点最大的信息熵

        for (int j = 0; j < midArray.size(); j++) {
            double gain = CalcInfoGain(midArray[j], dataSet, dataSet.Attribute[i]);
            if (gain >= maxmidEntrophy) {
                maxmidEntrophy = gain;
                bestmid = midArray[j];
            }
        }
        bestMid.push_back(bestmid); //将最优划分点放入vector容器
    }
    return bestMid;
}
```

9.CalcInfoGain()计算信息增益

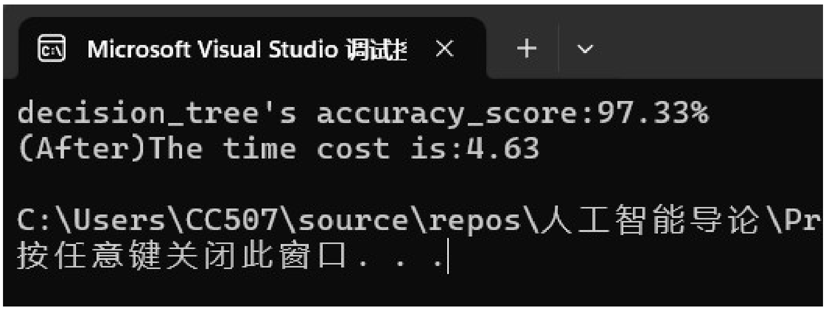
```cpp
double DecisionTree::CalcInfoGain(double midValue, DataSet& dataSet, string Value) { //计算一个属性的信息增益
    vector<vector<string>> leftData;
    vector<vector<string>> rightData;
    int leftCount = 0;
    int sum = dataSet.Data.size(); //样本数量
    double mid = midValue;

    for (int i = 0; i < dataSet.Data.size(); i++) {
        if (stod(dataSet.AttributeValue[Value][i]) <= mid) {
            leftCount++;
            leftData.push_back(dataSet.Data[i]);
        }
        else {
            rightData.push_back(dataSet.Data[i]);
        }
    }
    DataSet d1(leftData, dataSet.Attribute);
    DataSet d2(rightData, dataSet.Attribute);
    double gain = CalcEntropy(dataSet) - leftCount * CalcEntropy(d1) / sum - (sum - leftCount) * CalcEntropy(d2) / sum;

    return gain;
}
```

10.Main()函数

```cpp
int main()
{
    //开始计时
    clock_t start = clock();

    // 创建一个数据集对象
    DataSet dataSet;
    string fname = "D:\\learn\\大三下\\人工智能导论\\实验五决策树算法\\traindata.txt";
    Read_file(dataSet, fname);

    DecisionTree dt;
    Node* RootNode = new Node;

    //初始化父结点
    map<string, double> BestAttributeMap = dt.FindBestInfoGain(dataSet);
    string BestAttribute = BestAttributeMap.begin()->first;
    RootNode->Attribute = BestAttribute;

    vector<string> TempAttribute;
    TempAttribute.push_back(BestAttribute);
    vector<string> TempValue = dataSet.AttributeValue[BestAttribute];
    vector<string> TempAttr = dataSet.AttributeValue["类别"];
    vector<vector<string>> TempData(TempValue.size());
    for (int i = 0; i < TempValue.size(); i++) {
        TempData[i].push_back(TempValue[i]);
        TempData[i].push_back(TempAttr[i]);
    }
    DataSet d(TempData, TempAttribute);
    vector<double> MidArray = dt.FindMidValue(d);
    vector<double> Midd = dt.FindMidValue(dataSet);

    RootNode->MidValue = Midd;
    double bestmid = MidArray[0]; //最优属性的最优划分点
    RootNode->Mid = bestmid;
    RootNode->isRoot = true;

    // 生成决策树(训练)
    dt.TreeGenerate(dataSet, RootNode);

    DataSet dataSet_pre;
    string fname_pre;
    fname_pre = "D:\\learn\\大三下\\人工智能导论\\实验五决策树算法\\testdata.txt";
    Read_file(dataSet_pre, fname_pre);
    vector<string> pre;
    pre = dt.Prediction(dataSet_pre, RootNode);

    //计算正确率百分比
    double accuracy_score = 0;
    int count_accuracy = 0;
    for (int i = 0; i < pre.size(); i++) {
        if (pre[i] == dataSet_pre.AttributeValue["类别"][i]) count_accuracy++;
    }
    accuracy_score = count_accuracy * 1.0 / pre.size();
    cout << fixed << setprecision(2) << "decision_tree's accuracy_score:" << accuracy_sc


    // 释放内存
    dt.DestoryDecisionTree(RootNode);

    //运行时间
    clock_t finish = clock();
    cout << "(After)The time cost is:" << double(finish - start) / CLOCKS_PER_SEC << end

    return 0;
}
```

4 运行结果

使用训练数据集获得的决策树对新数据结果的预测成功率达到 97.33%



5 我的体会

通过本次实验，我掌握了决策树算法的核心概念，包括信息熵、信息增益、最优划分点等。通过深入理解算法原理、设计算法结构、编写代码实现、调试优化以及进行实验测试，我不仅加深了对决策树算法的理解，还提升了自己的编程能力和算法实现能力。这次实验充满了挑战和收获，让我对算法实现有了更深刻的认识，并为我的未来学习和工作打下了良好的基础。