ID3算法实现实验报告

范潇

2023年12月5日

1 实验目的

1. 掌握决策树算法（ID3）的基本思想
2. 掌握决策树算法实现的细节
3. 熟悉算法测试的基本流程

2 实验内容

1. 利用C++语言编写决策树算法（ID3）
2. 选取适当的数据集
3. 对选取的数据集进行预处理及划分
4. 利用验证集选取最佳的超参数
5. 对编写的算法进行5折交叉验证

3 实验过程

3.1 程序编写

整个算法的具体实现分为5个部分：

1. 用于存储数据集的数据结构设计
2. 用于存储决策树的数据结构设计

3. csv文件读取函数

4. 决策树构造函数

5. 用于测试和打印结点信息的函数

3.1.1 用于存储数据集的数据结构设计

由于在构造决策树的递归过程中，用于决策的样本个数以及属性在不断减少，所以需要一个能够灵活变化的数据结构来存储输入的数据集。因为在C++中没有python的pandas库中所提供的DataFrame数据结构，所以需要自行设计所需的数据结构。

```
//用于存储数据集的相关数据结构
typedef enum {
    CON = 0,//连续型
    DIS,//离散型
    CLASS,//类别
}data_type;
typedef vector<double> dis_data;//存储离散型数据
typedef struct {//存储连续型数据
    vector<double> raw_data;//原始数据
    double sep;//连续型数据离散化后所得的分界点
}con_data;
typedef pair<con_data, dis_data> my_data;
typedef struct {//一列数据
    data_type type;//数据类型
    my_data data;//该列数据
}my_column;
class dataset {//数据集
private:
    vector<string> header;
    vector<my_column> column;
    int attr_num;//属性个数
    int data_num;//样本个数
    double entropy;//熵
    int class_col;//类别所在列
public:
//未列出成员函数
};
```

其中，数值型数据离散化的伪代码如下：

Algorithm 1 数值型数据离散化

输入: 待离散化的一列数值型数据data

输出: 分界点sep

1: **function** DISCRITIZATION(vector<double> data)
2: 将data中的数据按照升序存储进辅助数组temp中
3: 将temp中每组相邻元素的平均数依次存放进辅助数组candidate中
4: 将candidate中的元素依次取出，并计算以此为分界点时所得到的两个子数据集的熵值之和
5: **return** 使得熵值之和最小的元素
6: **end function**

3.1.2 用于存储决策树的数据结构设计

```
//决策树结点
struct DT {
    data_type type;//存储的数据类型
    double sep;//存储连续型属性的分界点
    vector<double> val;//存储离散型属性的可能取值
    string header;//属性名称
    int ch_num;//孩子节点个数
    DT** children;//孩子结点数组
    double result;//存储的是叶子结点中的类别
    double entropy;//熵
    int data_num;//样本个数
};
//决策树
class DecisionTree {
public:
    DT* DTree;
    DecisionTree();
    void construct(DT*& root,dataset& source,double result_for_empty_data,
        int first);
    void test(dataset& test_set);
    void print(DT*,int );
    friend dataset;
};
```

3.1.3 csv文件读取函数

该函数要求用户依次输入训练集和测试集的文件名，然后输入阈值，最后会分别打印出训练集和测试集中各列的标题，并要求用户对每一列的数据进行分类。由于ID3决策树算法在处理连续型属性之前，需要先将其离散化，因此要求用户把数值属性用“C”标记出来。同时，要求用户把标称属性、类别、不参与决策树构造的属性分别用“D”、“T”、“X”标记出来。

3.2 决策树构造函数

Algorithm 2 ID3算法构造决策树

输入: 根节点root，数据集source

```
1: function CONSTRUCT(DT*& root,dataset& source)
2:     if source中无样本 then
3:         root→children =NULL
4:         root→result置为原始数据集中类别占比最多的一类
5:     else
6:         if source中无属性 then
7:             root→children =NULL
8:             root→result置为当前数据集中类别占比最多的一类
9:         else
10:            求最大信息增益
11:            if 不超过阈值 then
12:                root→children =NULL
13:                root→result置为当前数据集中类别占比最多的一类
14:            else
15:                将当前数据集按照信息增益最多的那个属性进行划分，然后递归
16:            end if
17:        end if
18:    end if
19: end function
```

3.3 用于测试和打印结点信息的函数

测试函数要求的输入是测试集。对于测试集中的每一个样本，它将按照其属性值，在已经构造好的决策树中搜索叶子结点，并将得到的值与测试集中对应样本的类别进行比较。

打印结点的函数采用的是横向打印，对于每个非叶子结点，将会输出对应数据表的熵、其中的样本数、以及能使信息增益最大的属性的信息。对于叶子节点，则会打印其对应的类别、样本数、以及熵。

3.4 数据集的选择

在挑选用于训练ID3决策树算法的数据集时，我所采用的标准是：1.数据规模较小；2.属性个数较少；3.即有标称属性，又有数值属性；4.缺失值较少。最终我选用了Kaggle平台上的Obesity Classification数据集。

3.5 数据集预处理和划分

我将“ID”这一列数据删去，并将“Gender”和“Label”这两列标称数据进行独热编码，其中“偏瘦”、“正常”、“偏胖”、“肥胖”分别被编码为0、1、2、3。然后将该数据集按照8：2的比例划分为训练集和验证集，最后再将它按照5折交叉验证的要求进行划分。

3.6 超参数的选取

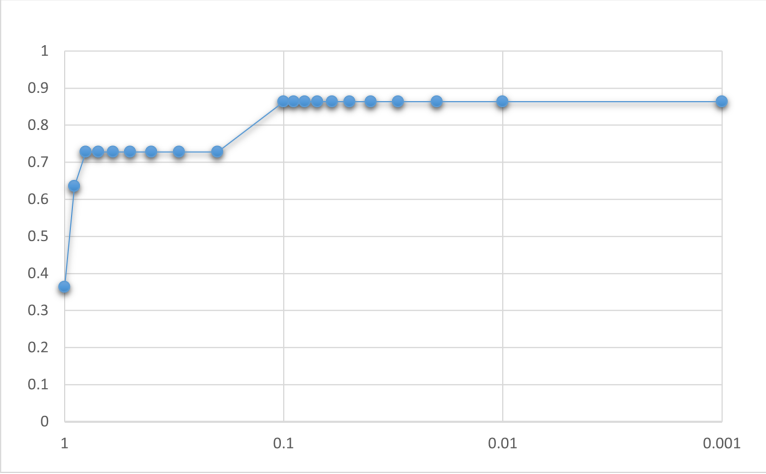

图 1: 超参数与准确率的关系

经过测试，最终将超参数选取为0.05.

4 测试样例说明

原数据集共有108份样例，有“ID”、“Age”、“Gender”、“Height”、“Weight”、“BMI”、“Label”六列数据。其中“ID”不参与决策树的构造，“Age”、“Gender”、“Height”、“Weight”、“BMI”为数值型数据，“Gender”、“Label”为标称型数据。“Label”即肥胖程度，是该数据集中的样本的类别，总有4类：“Underweight”、“Normal Weight”、“Overweight”、“Obese”。

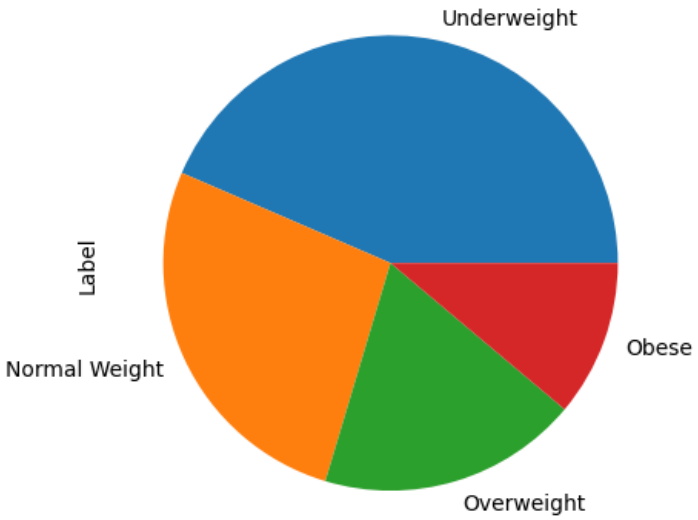

图 2: Label的分布情况

5 测试结果及分析

经过调参后，程序在验证集上的最高准确率达到86.3636%。当超参数，即阈值，选为0.05时，在5折交叉验证中的准确率分别为86.3636%、68.1818%、90.9091%、85.7143%、71.4286%，平均准确率为80.5195%。

以准确率为90.9091%的实验中生产的决策树为例，根据该决策树最多只需进行两次判断即可结束分类，第一次根据体重进行分类，如果相对较

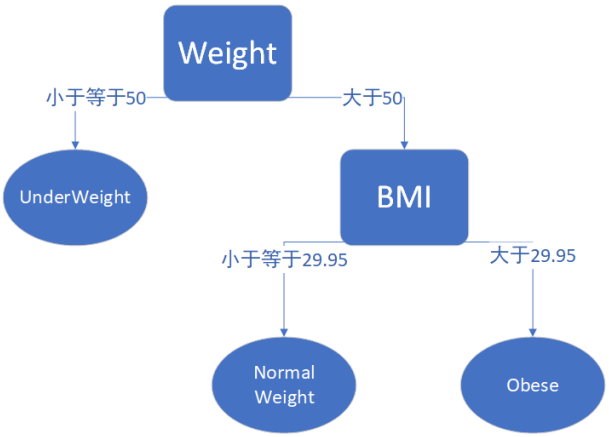

图 3: 算法生成的决策树

轻，则分类为“Underweight”，否则继续比较BMI。如果BMI又较大，则分为“Obese”，否则分为”Normal Weight”。这两次比较的依据都十分符合常识，因为医学上判断是否肥胖的指标无外乎体重和BMI。之所以该决策树并没有“Overweight”对应的叶子结点，是因为没有达到相应的阈值。而在这种情况下仍能够达到较高的准确率的原因之一是测试集样本的分布不均匀：在该测试集中，只有一个“Overweight”的样本。但是由于使用了K折交叉验证，由于测试集中样本的分布而导致的误差已被抵消掉一部分了。