基于ID3算法的决策树

姓名：季马泽宇

专业：自动化

学号：1950089

实验目的

了解决策树的基本思想，尝试通过代码简单实现决策树，使用一个小的数据集对算法进行测试。

决策树(Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。

决策树的算法有很多，主要有基于GINI指数和信息熵两种构建方式。这里我们使用ID3算法。

什么是信息熵

$H_i = -\sum_{k=1,P_{i,k}!=0}^{n} P_{i,k}log(P_{i,k})$
信息熵就是含的信息量，如果一个节点只有一类，那log1=0，即信息熵为0。
众所周知，熵的意思是无序程度，熵越大不确定度越高，我们总是希望不确定性低一点，所以熵要尽可能小。

ID3算法

最开始就一个节点，信息熵很大。我们使用某个特征对节点进行拆分，拆分完后几个节点的加权信息熵要尽可能小。信息增益就是原本的信息熵减掉新的信息熵（加权信息熵，节点信息熵×节点占父节点样本的个数），信息增益要尽可能大。

对于信息熵的一些理解我写在这篇博客里了(70条消息) 对信息熵的理解_季马宝宝的博客-CSDN博客，其他一些算法也可以参考我之前写的博客(69条消息) Machine Learning（Lesson4 决策树与集成学习）_季马宝宝的博客-CSDN博客

实验内容

- 实现了基于ID3算法的决策树
  - 通过计算信息增益递归生成决策树
  - 通过后剪枝提高泛化能力
- 实现了一个简单的DataFrame工具，用于读取数据与数据的简单处理
  - 提供从csv文件中读取数据，并自动推断数据类型，将每列生成一个Series，并组成DataFrame
  - one-hot编码，将字符串类型Series转为n(不同值个数)个0-1Series

- 将数值类型的特征组合成矩阵数组输出，供后续模型使用

实验过程

1. 通过定义的DataFrame类读入CSV数据（这里是类的定义和主函数中一部分，具体看提交代码实现）

```
//Series
template <typename T>
class Series
{
private:
   string feature_name; //name of feature
   vector<T> data;
   friend class DataFrame;

public:
   Series() = default;
   Series(string name) : feature_name(name){};
   Series(string name, vector<T> array) : feature_name(name), data(array){};
   ~Series(){};
   T &operator[](const int &);
   void append(T);
   int size() { return data.size(); }
   void set_feature_name(string name) { feature_name = name; }
   string get_feature_name() { return feature_name; }
   vector<T> get_data() { return data; }
};
//DataFrame
class DataFrame
{
private:
   int totol_data_numbers;
   vector<string> feature_names;
   map<string, pair<string, int>> name_to_pos; //using feature names to find
series
   vector<Series<int>> int_arrays;
   vector<Series<double>> double_arrays;
   vector<Series<string>> string_arrays;
public:
   DataFrame() { totol_data_numbers = 0; }
   ~DataFrame(){};
   //add cols
   void add_feature(Series<int>);
   void add_feature(Series<double>);
   void add_feature(Series<string>);
   //append data
   void append(meta_element element);
   vector<string> get_all_features() { return feature_names; }; //return all
feature names
   vector<string> get_all_numerical_features(string label_name); //return
all numerical feature names
   void display_datas();                                          //print
datas
   int get_data_numbers() { return totol_data_numbers; }        //return
numbers of datas
```

```
    vector<single_data> numerical_dataset(string);                //get
numerical dataset
};
//main
DataFrame df_train=read_csv("train.csv");//读入训练数据
df_train.display_datas();//打印数据
```

通过display_data函数可以打印出数据，可以看到字符串数据被转为了one-hot编码

Survived	:0	1	1	1	0	0	0	0	1	1
Pclass	:3	1	3	1	3	3	1	3	3	2
Sex	:male	female	female	female	male	male	male	male	female	female
Sex_female	:0	1	1	1	0	0	0	0	1	1
Sex_male	:1	0	0	0	1	1	1	1	0	0
Age	:22	38	26	35	35	30	54	2	27	14
SibSp	:1	1	0	1	0	0	0	3	0	1
Parch	:0	0	0	0	0	0	0	1	2	0
Fare	:7.3	71.3	7.9	53.1	8.1	8.5	51.9	21.1	11.1	30.1
Embarked	:S	C	S	S	S	Q	S	S	S	C
Embarked_C	:0	1	0	0	0	0	0	0	0	1
Embarked_Q	:0	0	0	0	0	1	0	0	0	0
Embarked_S	:1	0	1	1	1	0	1	1	1	0

2. 训练决策树

我采用的是二分决策树的策略，DataFrame工具会将所有的数值数据转为浮点型数据，输出一个矩阵数组。每次计算节点最大信息增益时，先枚举用于分类的数据，然后对数据根据此特征进行排序，然后依次将阈值设定为两个相邻不同数据之间，一次扫描即可快速求出最大的信息增益

Core Codes:

```
//计算确定特征下的最大信息增益
double CalculateGain(vector<single_data> DataSet,string feature,double
entropy,double &threshold,double &LeftEntropy,double &RightEntropy)
{
    map<int,int> LabelMap;//统计不同labels包含的个数
    int total=DataSet.size();//节点包含数据总数
    for(auto data:DataSet)
    {
        LabelMap[data.label]++;
    }
    // //函数内定义的比较函数,利用C++11新特性
    auto comp = [&](single_data A,single_data B){return
A.feature_to_value[feature]<B.feature_to_value[feature];};
    sort(DataSet.begin(),DataSet.end(),comp);//通过feature进行排序
    map<int,int> TmpLabelMap;//随着阈值更新，实时记录labels个数，扫一遍计算出信息熵
    double MinEntropy=1e6;
    int LeftNums=0;
    int RightNums=0;
    for(int i=1;i<DataSet.size();i++)//因为是排序后的，循环的其实是threshold，理论
上应该二分找不同的值，偷懒了
    {
        single_data data=DataSet[i];
        TmpLabelMap[data.label]++;
        if(abs(data.feature_to_value[feature]-DataSet[i-
1].feature_to_value[feature])<1e-6)continue;
        double TmpLeftEntropy=0;
        double TmpRightEntropy=0;
        for(auto lb:TmpLabelMap)//因为是二分决策树，一个节点分为两部分，更新阈值左边
每种标签的个数
        {
            double left=lb.second;
            double right=LabelMap[lb.first]-lb.second;
            double p_left=1.0*left/i; //分到左侧的该label下的数据占左侧比例
```

```
            double p_right=1.0*right/(total-i);//分到右侧的该label下的数据占右侧
比例
            TmpLeftEntropy=-p_left*(log2(p_left));
            TmpRightEntropy=-p_right*(log2(p_right));

        }
        if(feature=="Age"&&total<100)
        {
            //cout<<1;
        }
        double weight1=1.0*i/total;//左子树权重
        double weight2=1.0*(total-i)/total;//右子树权重
        if(weight1*TmpLeftEntropy+weight2*TmpRightEntropy<MinEntropy)
        {
            LeftNums=i;
            RightNums=total-i;
            MinEntropy=weight1*TmpLeftEntropy+weight2*TmpRightEntropy;
            RightEntropy=TmpRightEntropy;
            LeftEntropy=TmpLeftEntropy;
            threshold=data.feature_to_value[feature]-0.000001;//防止精度损失
        }
    }
    double gain=entropy-MinEntropy;
    return gain;
}
```

过程中，我让程序输出了每次划分的过程，包括使用什么数据进行划分，左节点划分多少数据，右节点划分多少数据，划分阈值以及信息增益

```
----------
depth:0
feature:Sex_male
threshold:0.999999
leftson:38
rightson:60
gain:0.607355
------------------------------------------------------------
------------------------------------------------------------
depth:1
feature:Embarked_S
threshold:0.999999
leftson:13
rightson:25
gain:0.0203269
------------------------------------------------------------
------------------------------------------------------------
depth:2
feature:Age
threshold:16
leftson:4
rightson:21
gain:0.0472643
------------------------------------------------------------
------------------------------------------------------------
depth:3
feature:Pclass
threshold:3
leftson:1
```

3. 剪枝

在决策树训练完成后，使用后剪枝进行，这里后剪枝的方式是在训练前留出一部分数据，在训练完成的决策树上有这些数据对节点进行测试，分别计算合并与不合并的信息熵，将负效果的节点进行合并。

Core Codes：

```
//后剪枝
void PostPruning(TreeNode *fa,vector<single_data> DataSet)
{
    if(!fa->left)return;//已经是叶子节点了
    int LeftSons,RightSons;
    double LeftEntropy=0,RightEntropy=0;
    vector<single_data> DataSet1;
    vector<single_data> DataSet2;
    for(auto data:DataSet)
    {
        if(data.feature_to_value[fa->feature]<fa-
>threshold)DataSet1.push_back(data);
        else DataSet2.push_back(data);
    }
    //左节点信息熵
    {
        map<int,int> LabelMap;//统计不同labels包含的个数
        int total=DataSet1.size();//节点包含数据总数
        for(auto data:DataSet1)
        {
            LabelMap[data.label]++;
        }
        double TmpEntropy=0;
        for(auto lb:LabelMap)
        {
            double p=1.0*lb.second/total;
            TmpEntropy+=-p*log2(p);
        }
        LeftSons=total;
        LeftEntropy=TmpEntropy;
    }
    //右节点信息熵
    {
        map<int,int> LabelMap;//统计不同labels包含的个数
        int total=DataSet2.size();//节点包含数据总数
        for(auto data:DataSet2)
        {
            LabelMap[data.label]++;
        }
        double TmpEntropy=0;
        for(auto lb:LabelMap)
        {
            double p=1.0*lb.second/total;
            TmpEntropy+=-p*log2(p);
        }
        RightSons=total;
        RightEntropy=TmpEntropy;
    }
    int total=RightSons+LeftSons;
    double weight1=1.0*RightSons/total;
    double weight2=1.0*LeftSons/total;
```

```
    if(fa->entropy-weight1*LeftEntropy+weight2*RightEntropy<0)
    {
        cout<<"cut:"<<fa->entropy-
weight1*LeftEntropy+weight2*RightEntropy<<endl;
        //没释放内存
        fa->left=NULL;
        fa->right=NULL;
        return;
    }
    PostPruning(fa->left,DataSet1);
    PostPruning(fa->right,DataSet2);
}
```

测试样例说明

使用了Kaggle上经典的Titanic数据集进行演示，进行了一些简单的数据清洗，直接暴力把缺省值都删了，所以数据量不是太够，删除了存在字符串和数字组合的特征（比如船票，真实处理起来要手动分类，直接当数值处理也不妥）

train.csv

```
Survived,Pclass,Sex,Age,SibSp,Parch,Fare,Embarked
0,3,male,22,1,0,7.3,S
1,1,female,38,1,0,71.3,C
1,3,female,26,0,0,7.9,S
1,1,female,35,1,0,53.1,S
0,3,male,35,0,0,8.1,S
0,3,male,30,0,0,8.5,Q
0,1,male,54,0,0,51.9,S
0,3,male,2,3,1,21.1,S
1,3,female,27,0,2,11.1,S
1,2,female,14,1,0,30.1,C
1,3,female,4,1,1,16.7,S
1,1,female,58,0,0,26.6,S
0,3,male,20,0,0,8.1,S
0,3,male,39,1,5,31.3,S
0,3,female,14,0,0,7.9,S
1,2,female,55,0,0,16.0,S
0,3,male,2,4,1,29.1,Q
1,2,male,30,0,0,13.0,S
0,3,female,31,1,0,18.0,S
1,3,female,30,0,0,7.2,C
0,2,male,35,0,0,26.0,S
1,2,male,34,0,0,13.0,S
1,3,female,15,0,0,8.0,Q
1,1,male,28,0,0,35.5,S
0,3,female,8,3,1,21.1,S
1,3,female,38,1,5,31.4,S
0,3,male,30,0,0,7.2,C
0,1,male,19,3,2,263.0,S
```

将数据分成了三份，train.csv,cut.csv和test.csv,分别用于训练、剪枝和测试，大约是1:1:1

测试结果与分析

分别测试了几组数据，分别选择了不同的决策树最大深度以及节点的最小大小

```
tree.BuildTree(tree.root,1,3,1);//开始训练，决策树最大深度为3，最小节点大小为1；
tree.BuildTree(tree.root,1,5,1);//开始训练，决策树最大深度为5，最小节点大小为1；
tree.BuildTree(tree.root,1,3,5);//开始训练，决策树最大深度为3，最小节点大小为5；
tree.BuildTree(tree.root,1,5,5);//开始训练，决策树最大深度为5，最小节点大小为5；
vector<single_data> dataset2=df_cut.numerical_dataset(label);  //后剪枝
```

由于数据量有限，效果比较单薄

这是决策树最大深度为3时的结果，虽然经过了一个结点的后剪枝，约束节点大小没有区别，因为层数有限没有分出特别小的节点，但是剪枝前后没有太大区别。

```
------------------------------------------------------------
depth:0
feature:Sex_female
threshold:0.999999
leftson:67
rightson:46
gain:0.599074
------------------------------------------------------------
------------------------------------------------------------
depth:1
feature:Age
threshold:22
leftson:13
rightson:54
gain:0.233928
------------------------------------------------------------
------------------------------------------------------------
depth:1
feature:Embarked_S
threshold:0.999999
leftson:19
rightson:27
gain:0.0405738
------------------------------------------------------------
correct:87 wrong:19
cut:-0.0763654
correct:87 wrong:19
```

下面是五层决策树，如果限定节点大小为5与之前3层结果相同，因为产生不了那么小的节点。但是如果节点大小不限制，那么就会产生过拟合现象，可以看到在经过三次剪枝后，模型准确率有少量提升。

```
loading data 113 records
initial entropy:0.974942

------------------------------------------------------------
depth:1
feature:Sex_female
threshold:0.999999
leftson:67
rightson:46
gain:0.599074
------------------------------------------------------------
------------------------------------------------------------
depth:2
feature:Age
threshold:22
leftson:13
rightson:54
gain:0.233928
------------------------------------------------------------
------------------------------------------------------------
depth:3
feature:Fare
threshold:7.2
leftson:1
rightson:53
```

```
gain:0.000534514
------------------------------------------------------------
------------------------------------------------------------
depth:2
feature:Embarked_S
threshold:0.999999
leftson:19
rightson:27
gain:0.0405738
------------------------------------------------------------
------------------------------------------------------------
depth:3
feature:Age
threshold:38
leftson:15
rightson:4
gain:0.00836488
------------------------------------------------------------
------------------------------------------------------------
depth:4
feature:Age
threshold:49
leftson:2
rightson:2
gain:0.0612781
------------------------------------------------------------
------------------------------------------------------------
depth:3
feature:Age
threshold:16
leftson:4
rightson:23
gain:0.04132
------------------------------------------------------------
------------------------------------------------------------
depth:4
feature:Pclass
threshold:3
leftson:1
rightson:3
gain:0.103759
------------------------------------------------------------
------------------------------------------------------------
depth:4
feature:SibSp
threshold:0.999999
leftson:11
rightson:12
gain:0.0477614
------------------------------------------------------------
correct:85 wrong:21
cut:-0.0763654
cut:-0.288722
cut:-0.210527
correct:87 wrong:19
```