中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	**周四 7-8 节**	专业（方向）	**互联网**
学号	15352010	姓名	**蔡烨**

一、 实验题目

通过构建决策树，预测数据的标签

二、 实验内容

1. 算法原理

带有多个特征的数据集，可以通过对这些特征进行分类，层层分类之后得到一棵树。通过将测试集数据根据树上特征的分类，对应到不同的标签下，从而预测该数据的标签。而在建树的时候，选择哪一个特征来进行分类，成为最大的问题。

三种特征选择的方法：

（1）使用信息增益：ID3

两个离散变量 X 和 Y 的互信息 I(X;Y)衡量的是这两个之间的相关度。信息增益计算的便是特征 A 和数据集 D 之间的互信息，g(D,A) =I(D,A) = H(D) − H(D|A)。其中 $H(D) = -\sum_{d\in D} p(d)logp(d)$ 是数据集 D 的经验熵，特征 A 对数据集 D 的条件熵是 $H(D|A) = \sum_{a\in A} p(a)H(D|A=a)$. 信息增益最大时，条件熵最小，即在该特征的情况下，数据集标签的不确定性最小。选用信息增益最大的特征进行分类。

（2）使用信息增益率：C4.5

当出现某个特征的类别比较多时，该特征的不确定性会比较小，容易被选中进行分类，但是这种情况容易造成过拟合，因此，C4.5 采用信息增益率而不是信息增益。gRatio(D,A) = (H(D)-H(D|A))/SpiltInfo(D,A). 信息增益率就是拿算出来的信息增益除以数据集 D 关于特征 A 的值的熵 SpiltInfo(D,A).

（3）使用 GINI 指数：CART

基尼系数越小，表示不确定性越小。通过计算在某个特征的条件下，数据集 D 的基尼系数，从而选出基尼系数最小的那个特征进行分类。$\text{gini}(D,A) = \sum_{J=1}^{V} p(A_j) * gini(D_j|A=A_j)$ 其中 $\text{gini}(D_j|A=A_j) = \sum_{i=1}^{n} p_i(1-p_i) = 1-\sum_{i=1}^{n} {p_i}^2$

分割训练集和验证集

这次实验用了两种方法进行训练集和验证集的分割：

（1）将整个文件读出的所有数据，随机打乱之后，拿出前 3/4 作为训练集，后 1/4 作为验证集，然后用三种选择方法分别进行建树和预测。但是因为这种方法是不可还原的，因此在每一次划分完数据的同时，除了对验证集进行预测，也对测试集进行预测。最后将验证集的

预测准确率最高的那一次跑出来的测试集结果，作为测试集的预测结果。这种方法的缺点是，一整个数据集的预测结果是难以再次重现的。

（2）因为在数据中，前 324 行数据的标签都是 1，后面的都是-1。为了避免数据单一化，从前 324 行中选出连续的 150 行，同时从后面的数据中也选出连续的 150 行，300 行一起作为验证集。通过多次测试，确定连续数据的从哪一行开始选用时，验证集的准确率最高。使用这个时候的训练集训练的模型来对测试集进行预测。这种方法的缺点是，因为训练集的选取具有局限性（连续选用），因此结果往往难以达到足够优秀。

本次实验最后提交的预测结果，是选用对当次验证集预测准确率最高的时候预测的测试集结果。因为验证集的准确率并不是高的离谱，因此此处忽略模型的过拟合。

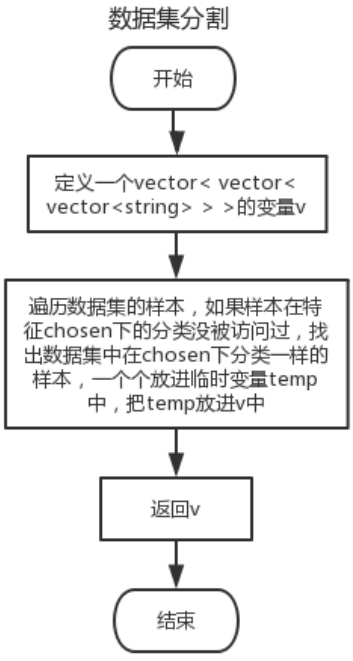
悲观剪枝 PEP

在建树之后，为了使树的泛化能力更强，对树进行后剪枝。从下往上遍历，比较每个节点剪枝前和剪枝后的误判率，选择误判率小的一种执行。**剪枝前**：算出当前节点为根节点的子树对当前数据集的误判率；**剪枝后**：将当前节点所在的子树砍掉，直接替换成一个叶子节点，叶子节点的标签由当前数据集中出现次数最多的标签决定，算出这种情况下的误判率。

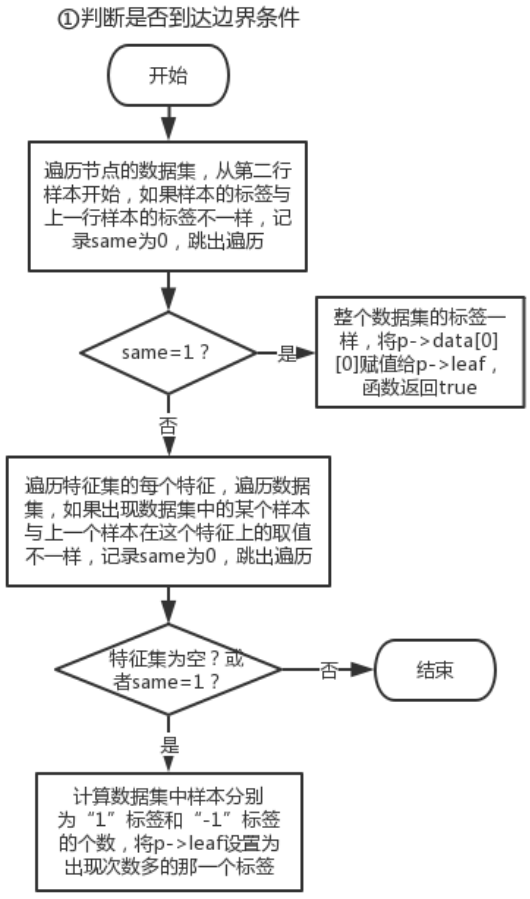
在计算误判率时，加入叶子惩罚机制。误判率=（误判样本个数+0.5*当前子树总的叶子个数）/ 当前数据集样本个数。

2. 流程图

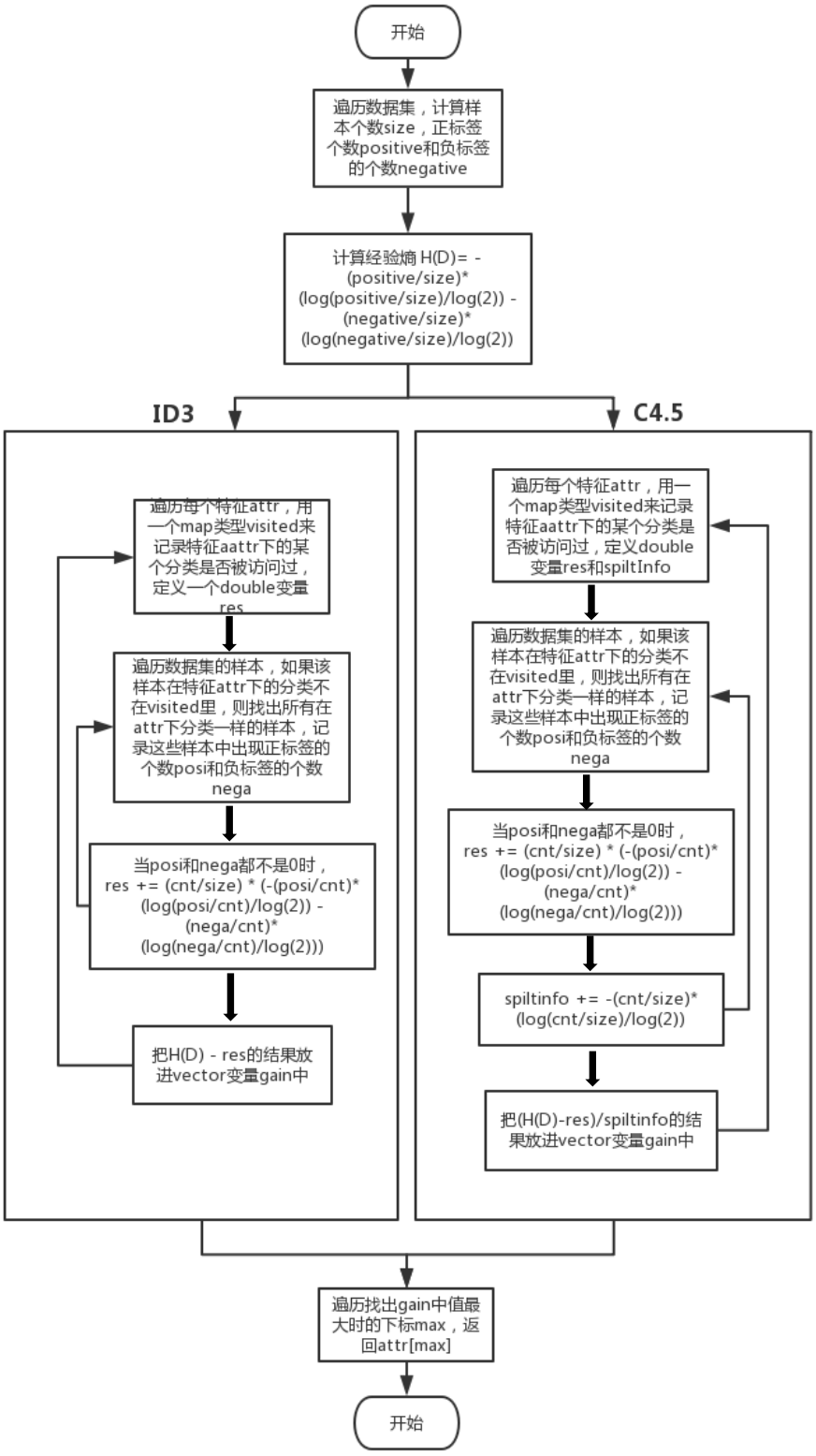
分割数据集：



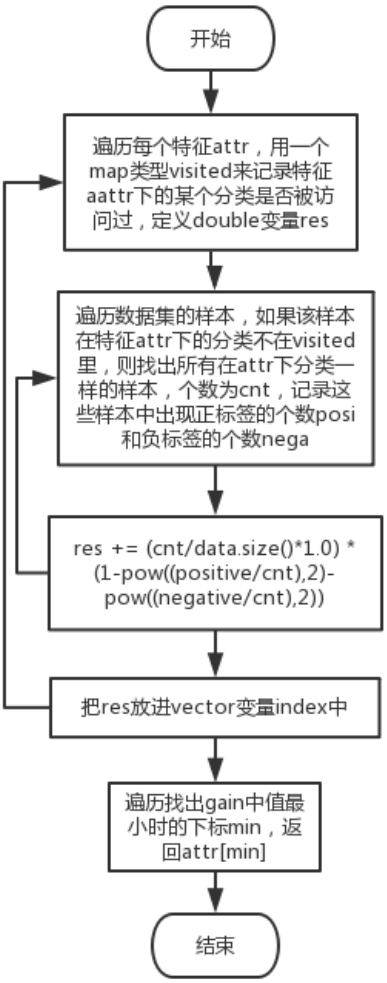
判断是否到达边界条件：



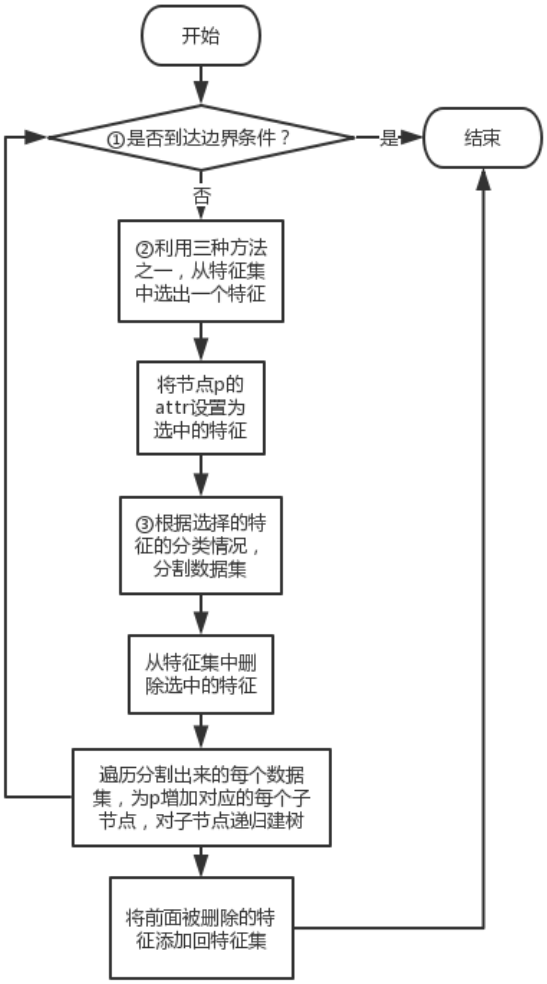
ID3 和 C4.5:


开始
遍历数据集，计算样本个数size，正标签个数positive和负标签的个数negative
计算经验熵 H(D)= -(positive/size)*(log(positive/size)/log(2)) -(negative/size)*(log(negative/size)/log(2))
ID3
C4.5
遍历每个特征attr，用一个map类型visited来记录特征aattr下的某个分类是否被访问过，定义一个double变量res
遍历数据集的样本，如果该样本在特征attr下的分类不在visited里，则找出所有在attr下分类一样的样本，记录这些样本中出现正标签的个数posi和负标签的个数nega
当posi和nega都不是0时，res += (cnt/size) * (-(posi/cnt)*(log(posi/cnt)/log(2)) -(nega/cnt)*(log(nega/cnt)/log(2)))
把H(D) - res的结果放进vector变量gain中
遍历每个特征attr，用一个map类型visited来记录特征aattr下的某个分类是否被访问过，定义double变量res和spiltInfo
遍历数据集的样本，如果该样本在特征attr下的分类不在visited里，则找出所有在attr下分类一样的样本，记录这些样本中出现正标签的个数posi和负标签的个数nega
当posi和nega都不是0时，res += (cnt/size) * (-(posi/cnt)*(log(posi/cnt)/log(2)) -(nega/cnt)*(log(nega/cnt)/log(2)))
spiltinfo += -(cnt/size)*(log(cnt/size)/log(2))
把(H(D)-res)/spiltinfo的结果放进vector变量gain中
遍历找出gain中值最大时的下标max，返回attr[max]
开始

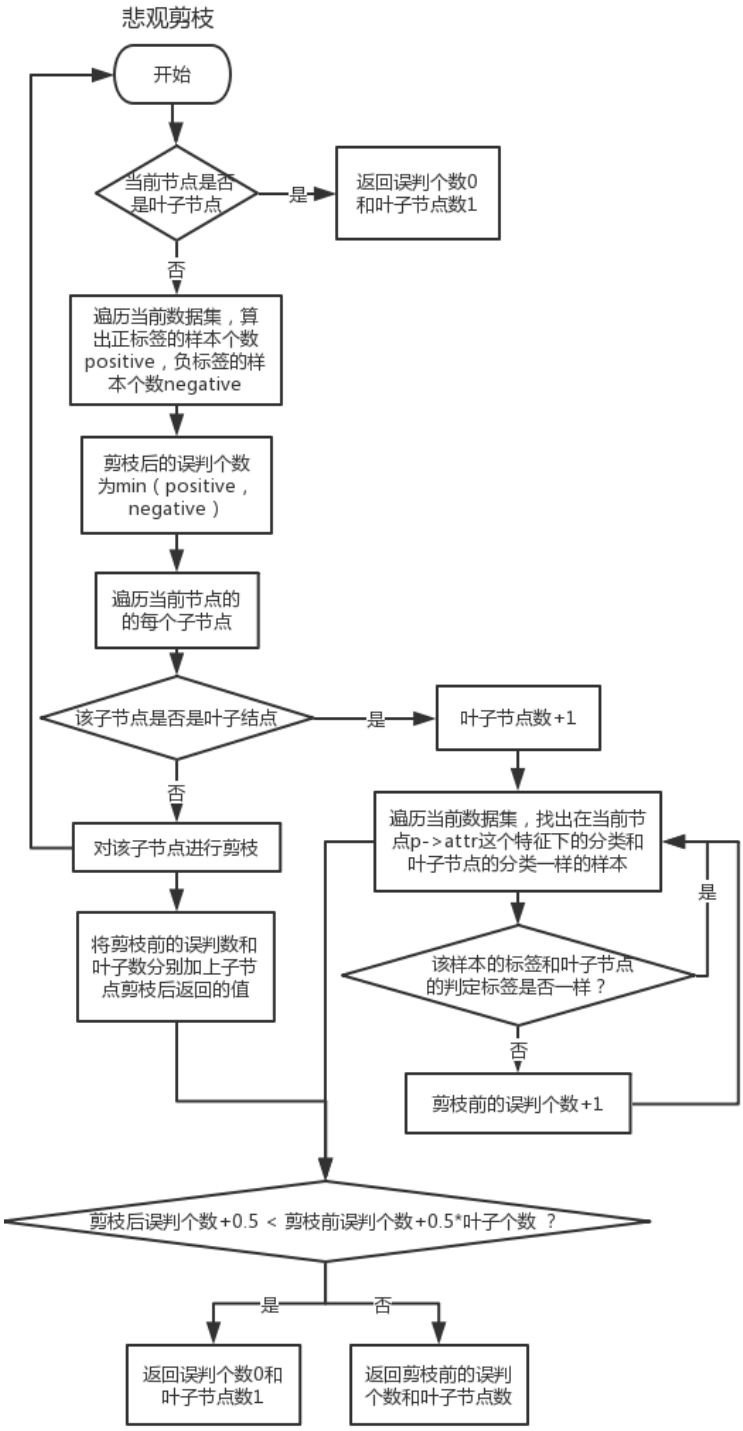
基于基尼系数的 CART:



递归建树



剪枝：



3. 关键代码截图（带注释）

ID3 的实现：

```
int ID_3(vector<vector<string>> data){
    double size = data.size() * 1.0;
    //计算经验熵 H(D)
    double positive = 0;
    for(int i=0; i<data.size(); i++)
        if(data[i][0] == "1")   positive += 1;
    double negative = size - positive;
    double H_D = -(positive/size)*(log(positive/size)/log(2)) - (negative/size)*(log(negative/size)/log(2));
    //计算每个特征下的条件熵
    vector<double> gain;//信息增益
    for(int i=0; i<attr.size(); i++){  //遍历每个属性
        double res = 0;
        map<string,bool> visited;
        for(int j=0; j<data.size(); j++){
            if(visited.count(data[j][attr[i]])) continue;//在特征attr[i]下的该分类已经被计算过
            visited[data[j][attr[i]]] = true;

            double cnt=0, posi=0, nega=0;
            for(int k=j; k<data.size(); k++){//找出所有和样本j在特征attr[i]下分类一样的样本
                if(data[j][attr[i]] == data[k][attr[i]]){
                    cnt += 1;
                    if(data[k][0] == "1")   posi +=1;
                    else if(data[k][0]=="-1")   nega += 1;
                }
            }
            if(posi!=0 && nega!=0)//为了避免log（0）出错
                res += (cnt/size) * (-(posi/cnt)*(log(posi/cnt)/log(2)) - (nega/cnt)*(log(nega/cnt)/log(2)));
        }
        gain.push_back(H_D - res);
    }
    int max = 0;
    for(int i=0; i<gain.size(); i++)
        if(gain[i] > gain[max])
            max = i;
    return attr[max];
}
```

C4.5 的实现：

```
int C4_5(vector<vector<string>> data){
    double size = data.size() * 1.0;
    //计算经验熵H(D)
    double positive = 0;
    for(int i=0; i<data.size(); i++)
        if(data[i][0] == "1")   positive += 1;
    double negative = size - positive;
    double H_D = -(positive/size)*(log(positive/size)/log(2)) - (negative/size)*(log(negative/size)/log(2));
    //计算每个特征下的条件熵
    vector<double> gain;//信息增益率
    for(int i=0; i<attr.size(); i++){ //遍历每个属性
        double res = 0;
        double spiltinfo = 0;
        map<string,bool> visited;
        for(int j=0; j<data.size(); j++){
            if(visited.count(data[j][attr[i]])) continue;
            visited[data[j][attr[i]]] = true;

            double cnt=0, posi=0, nega=0;
            for(int k=j; k<data.size(); k++){
                if(data[j][attr[i]] == data[k][attr[i]]){
                    cnt += 1;
                    if(data[k][0] == "1")   posi +=1;
                    else if(data[k][0]=="-1")   nega += 1;
                }
            }
            if(posi!=0 && nega!=0)
                res += (cnt/size) * (-(posi/cnt)*(log(posi/cnt)/log(2)) - (nega/cnt)*(log(nega/cnt)/log(2)));
            spiltinfo += -(cnt/size)*(log(cnt/size)/log(2));//比ID3多的唯一一步

        }
        gain.push_back((H_D-res)/spiltinfo);
    }
    int max = 0;
    for(int i=0; i<gain.size(); i++)
        if(gain[i] > gain[max])
            max = i;
    return attr[max];
```

CART 的实现：

```
int CART_gini(vector<vector<string>> data){
    vector<double> index;
    //计算每种特征的基尼系数
    for(int i=0; i<attr.size(); i++){ //遍历每个属性
        double res = 0;
        map<string,bool> visited;
        for(int j=0; j<data.size(); j++){//遍历数据集
            if(visited.count(data[j][attr[i]])) continue;//在特征attr[i]下的该分类已经被计算过
            visited[data[j][attr[i]]] = true;

            double cnt=0, positive=0, negative=0;
            for(int k=j; k<data.size(); k++){//找出数据集中所有和样本j在特征attr[i]下分类一样的样本
                if(data[j][attr[i]] == data[k][attr[i]]){
                    cnt += 1;
                    if(data[k][0] == "1")   positive += 1;
                    else if(data[k][0] == "-1") negative += 1;
                }
            }
            //p(a)*[1-sum(pi^2)]
            res += (cnt/data.size()*1.0) * (1-pow((positive/cnt),2)-pow((negative/cnt),2));
        }
        index.push_back(res);
    }
    int min = 0;
    for(int i=1; i<index.size(); i++){
        if(index[i] < index[min])
            min = i;
    }
    return attr[min];
}
```

判断是否到达边界条件的实现：

```
bool meet_with_bound(node *p)
{   //所有样本属于同一类别
    bool same = 1;
    for(int i=1; i<p->data.size(); i++){
        if(p->data[i][0] != p->data[i-1][0]){
            same = 0;
            break;
        }
    }
    if(same){
        p->leaf = p->data[0][0];
        return 1;
    }
    //在特征集中所有特征上取值相同，或者特征集为空，无法再分，取出现次数多的作为Label
    same = 1;
    for(int i=0; i<attr.size(); i++){
        for(int j=1; j<p->data.size(); j++){
            if(p->data[j][attr[i]] != p->data[j-1][attr[i]]){
                same = 0;
                break;
            }
        }
        if(!same)   break;
    }
    if(attr.size()==0 || same){
        int posi = 0;
        for(int i=0; i<p->data.size(); i++){
            if(p->data[i][0] == "1")
                posi++;
        }
        if(posi >= (p->data.size()-posi))
            p->leaf = "1";
        else
            p->leaf = "-1";
        return 1;
    }
    return 0;
```

数据集分割：

```
//数据集分割
vector<vector<vector<string>>> divide_data(vector<vector<string>> D, int chosen)
{
    vector<vector<vector<string>>> v;
    map<string,int> visited;
    for(int i=0; i<D.size(); i++){
        if(visited.count(D[i][chosen])) continue;//该分类已经被计算过
        visited[D[i][chosen]] = 1;
        vector<vector<string>> temp;
        for(int k=i; k<D.size(); k++){
            if(D[k][chosen] == D[i][chosen]){//在特征chosen下的分类相同
                vector<string> line;
                line = D[k];
                temp.push_back(line);
            }
        }
        v.push_back(temp);
    }
    return v;
}
```

递归建树：

```
//递归建树
void recursive(node *p,int method)
{
    if(meet_with_bound(p)){ return; } //边界条件
    int attr_chosen = choose_attr(p->data,method);//method= 1:ID3; 2:C4.5; 3:CART
    p->attr = attr_chosen; //选中的特征
    vector<vector<vector<string>>> subsets = divide_data(p->data, attr_chosen);//分割数据集
    //删除选中的特征
    vector<int>::iterator it = attr.begin();
    while((*it)!= attr_chosen)  it++;
    attr.erase(it);
    //将每一个分类添加为孩子节点
    for(int i=0; i<subsets.size(); i++){
        node *subnode = new node;
        subnode->data = subsets[i];
        p->children.push_back(subnode);
        subnode->_class = subsets[i][0][attr_chosen];
        //对孩子节点进行递归建树
        recursive(subnode,method);
    }
    //添加回前面被删掉的特征
    it = attr.begin();
    while((*it) < attr_chosen)  it++;
    attr.insert(it,attr_chosen);
    return;
}
```

剪枝

```
//剪枝
//返回的是当前子树的错误个数和叶子节点数
pair<int,int> cutLeaf(node *p){
    if(p->leaf != "0"){
        return make_pair(0,1);
    }
    //剪枝后误判个数
    double positive=0, negative=0, errorAfter=0, errorBefore=0, leave=0;
    for(int i=0; i<p->data.size(); i++){
        if(p->data[i][0] == "1")    positive+=1;
    }
    negative = p->data.size()-positive;
    errorAfter = (positive >= negative) ? negative : positive;
    //没剪枝时的误判个数和叶子节点
    for(int i=0; i<p->children.size(); i++){
        if(p->children[i]->leaf == "0"){//非叶子节点
            pair<int,int> temp = cutLeaf(p->children[i]);
            errorBefore += temp.first;
            leave += temp.second;
        }
        else{//叶子节点
            leave += 1;
            for(int j=0; j<p->data.size(); j++){
                if(p->data[j][p->attr] == p->children[i]->_class)//样本属于该叶子节点的分类
                    if(p->data[j][0] != p->children[i]->leaf)
                        errorBefore += 1;
            }
        }
    }
    if((errorAfter+0.5) < (errorBefore+0.5*leave)){//剪枝后错误率小
        p->leaf = (positive > negative) ? "1" : "-1";
        return make_pair(errorAfter,1);
    }
    else//不剪枝
        return make_pair(errorBefore,leave);
}
```

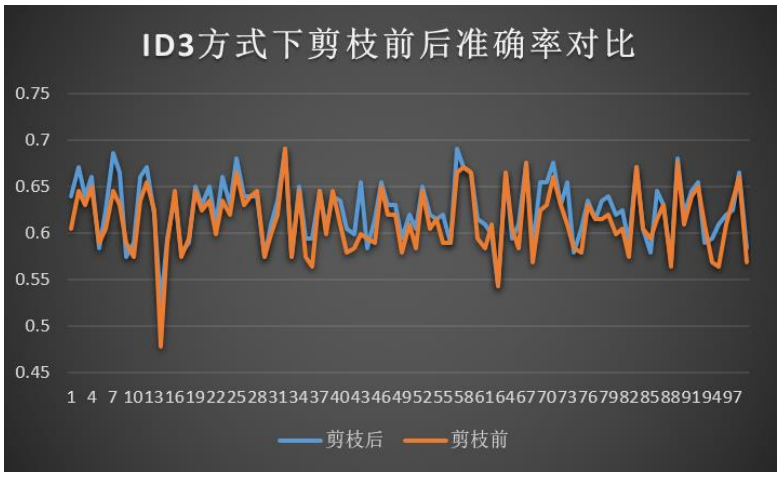
4. 创新点&优化（如果有）

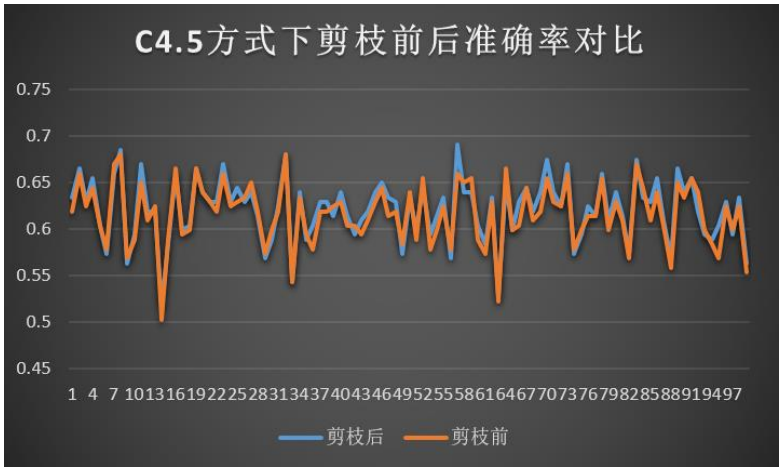
（1）随机选取训练集和验证集进行模型训练，详见实验原理部分

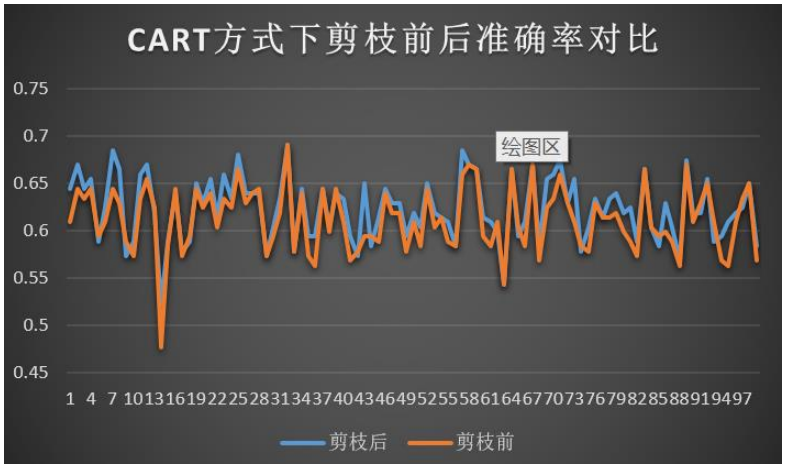
（2）悲观剪枝，详见实验原理、流程图及代码截图。

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）







不同特征选择方式下剪枝前后的平均准确率（100 次随机训练集）

特征选择方式	ID3	C4.5	CART
剪枝前	0.613342	0.61652	0.612111
剪枝后	0.624776	0.621238	0.62334

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

小数据集测试：

train_small：　　test_small:　　result_small:

train_small.csv

```
1,1,1
1,1,1
1,0,-1
0,0,1
0,1,-1
0,1,-1
0,0,-1
1,0,1
0,1,-1
```

test_small.csv

```
1,0,?
0,0,?
0,1,?
```

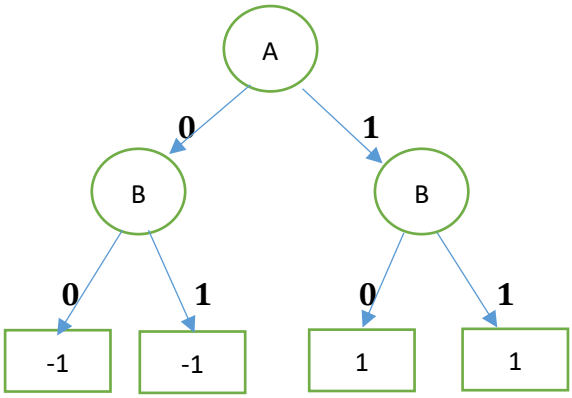
cut_small.csv

```
1
-1
-1
```

该数据集的经验熵为 H(D)=0.991

gain(A,D)=0.229　gain(B,D)=0.007

因此选用特征 A 作为根节点，构建出来的树是这样的：



测试集的测试结果，也是正确的。

3. 思考题

（1）决策树有哪些避免过拟合的方法？

答：扩大数据集，让模型拥有更多的范例；使用剪枝或者决策树深度限制等，避免决策树过于拟合训练集。

（2）C4.5 相比于 ID3 的优点是什么？

答：使用 ID3 时，有可能出现某个特征的分类非常多，每一个分类下的样本比较少，从而导致，只要根据该特征来进行分类，样本的误判率就会降低，不确定会比较小。但是这样容易造成过拟合，因此采用 C4.5。

（3）如何用决策树来判断特征的重要性？

答：决策树通过判断基于某个特征划分出来的样本相关度来判断该特征的重要性。ID3 算法是用数据的经验熵与特征的条件熵的差值作为信息增益，信息增益越大，该特征与数据的相关性越大，特征越重要。C4.5 在 ID3 的基础上，用信息增益除以特征本身的熵，成为信息增益率，避免了 ID3 算法下倾向于选择划分种类多的特征。CART 算法则是通过计算特征条件下的数据集的基尼指数来判断数据集的不确定性。基尼指数越小，不确定性越小，特征越重要。

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想