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	**m2**	专业（方向）	**软件工程（移动工程信息）**
学号	**15352446**	姓名	**钟展辉**

一、 实验题目

决策树（Decision Tree）

二、 实验内容

1. 算法原理

决策树：决策树是一个通过训练的数据来搭建起的树结构模型，根节点中存储着所有数据集和特征集，当前节点的每个分支是该节点在相应的特征值上的表现，而叶子节点存放的结果则是决策结果。通过这个模型，我们可以高效的对于未知的数据进行归纳分类。每次使用决策树时，是将测试样本从根节点开始，选择特征分支一直向下直至到达叶子节点，然后得到叶子节点的决策结果。

其中决策树算法的关键在于如何选出一个特征，根据这个特征对数据集进行分裂，随着划分的不断进行，希望决策树的分支节点所包含的样本尽可能的属于同一类别。刚开始时特征集中有很多特征，其中有的特征十分重要而有的特征无足轻重，我们倾向于先选择重要的特征对根节点进行分支，因为如果前面的节点就使用意义不大的特征进行分支，则最终使用这个决策树预测结果时，没走几步直接就进入了错误的子节点中，这样再向下遍历也没有意义。而关于本如何选择具有最优划分属性的特征，有以下的几种方法。

3 个特征选择方法：

这几种方法有涉及到熵的概念，在信息论里，熵是对不确定性的测量，假如一个随机变量 X 的取值为{x1,x2,x3...}，每一种取到的概率分别是{p1,p2,p3...}，那么 X 的熵定义为 $H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$.意思是一个变量的变化情况可能越多，那么它携带的信息量就越大。

然后是信息增益的概念，信息增益是针对特征而言的，就是看一个特征，系统有它和没有它时的信息量各是多少，两者的差值就是这个特征给系统带来的信息量，即信息增益。在本次实验中，信息增益就是决策树在进行特征选择划分前和划分后信息量的差值。

使用信息增益：ID3

（1）经验熵：Entropy(S),用当前数据集 S 以及式 $$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$ 计算得出的熵，代表当前数据集 S 的有序化程度。

（2）条件熵：Entropy(S|T)，使用特征 T 将数据集 S 划分成几部分后，分别计算他们的熵，然后按百分占比相加得出的熵，代表使用特征 T 划分后，当前数据集 S 的有序化程度。

（3）信息增益：IG(T)=Entropy(S)-Entropy(S|T)，以经验熵减去条件熵，就知道使用这个特征后能带来多少信息增益。

ID3 算法的核心思想：在决策树的每一个非叶子结点划分之前，先计算每一个特征所带来的信息增益，选择其中最大信息增益的特征来划分该节点，因为信息增益越大，区分样本的能力就越强，越具有代表性，

使用信息增益率：C4.5

C4.5 算法是对 ID3 算法的改进，ID3 算法是用信息增益来选择特征的，而信息增益的缺点是比较偏向选择取值较多的特征，如果有的特征取值比其他特征的取值多很多的话，这个特征基本就直接被认为是最重要的特征，但实际却未必。因此在 C4.5 中，为了改善这种情况，引入了对取值数量的惩罚，得到信息增益率作为特征重要性的衡量标准。如果某一个特征取值越多，那么对他的惩罚越严重，其信息增益率也能得到控制。

（1）每个特征的信息增益 IG(T)

（2）每个特征在这个数据集中的熵 IV(T)，即每个特征有多种取值，那么应用公式 $$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$ 即可求出每个特征的熵

（3）信息增益率 IGR=IG(T)/IV(T),特征的熵作为分母，如果某个特征可能的取值很多，则混乱程度很高，熵很大，会削弱这个特征的信息增益率。

使用 GINI 系数：CART

GINI 系数

CART 假设决策树是二叉树，内部结点特征的取值为“是”和“否”，左分支是取值为“是”的分支，右分支是取值为“否”的分支。在 CART 算法中,GINI 系数表示一个随机选中的样本在子集中被分错的可能性。GINI 系数为这个样本被选中的概率乘以它被分错的概率。

$$gini(D_j|A = A_j) = \sum_{i=1}^{n} p_i(1 - p_i) = 1 - \sum_{i=1}^{n} p_i^2$$

gini 系数值越小表示不确定性越小，当一个节点中所有样本都是一个类时，GINI 系数为零。我们选取 gini 系数最小的特征作为决策点

2. 伪代码

```
double Empirical_entropy(Node *p){     return 当前数据集的经验熵 HD;     }
double Conditional_entropy(int attr_index,Node *p){
    return 索引位置为 attr_index 的特征的条件熵 HDA ;
}
int ID3(Node *p){
    double HD=Empirical_entropy(p);//经验熵
    double HDA[10];//条件熵
    for i=0:p->attrsize:
        HDA[i]=Conditional_entropy(i,p);
        G[i]=HD-HDA[i]//信息增益
    return 信息增益最大的特征;
}
double SplitInfo(int attr_index,Node* p){
    return 索引位置为 attr_index 的特征的熵 ;
}
int C4_5(Node* p){
    double HD=Empirical_entropy(p);
    for i=0:p->attrsize:
        HDA=Conditional_entropy(i,p);//条件熵
        double sp_info=SplitInfo(i,p);//特征的熵
        gainRatio[i]=(HD-HDA)/sp_info;//信息增益率
    return 信息增益率最大的特征;
}
double gini(int attr_index,Node* p){
    return 索引位置为 attr_index 的特征的 gini 系数 ;
}
int CART(Node* p){
    double Gini[10];
    for i=0:p->attrsize:
        Gini[i]=gini(i,p);
    return 基尼系数最小的特征;
}
int choose_attr(Node *p){
    int attr;//被选中的特征，下面有三种选择方法
    attr=ID3(p);
    //attr=C4_5(p);
    //attr=CART(p);
    return attr;
}
void decide_final_label(Node* p){
    多数投票，在数据集中    正标签多 return 1;     负标签多 return -1;   }
```

```
bool meet_with_bound(Node* p){
    if 符合边界条件一：所有数据集样本的标签都相同; return 1;
    if 边界条件二：所有数据集的每个特征的取值都相同; return 1;
    else return 0;//即继续向下分支
}
void recursive(Node *p){
    if(meet_with_bound(p)){
        decide_final_label(p);//决定这个叶子结点的标签
        return;
    }
    未达到边界，则继续向下分支
    int attr_chosen=choose_attr(p);//选择一个当前特征集中最优代表性的特征
    int index_chosen;
    找出这个被选中的特征在特征集里的索引位置即 index_chosen;
    int maxnum,minnum;
    分别求出特征取值的最大最小值;
    for i=minnum:maxnum :
        Node *tem=new Node;
        新节点的特征集是父节点特征集除去被选中的哪一个特征后所剩的特征集;
        新节点的数据集是父节点被选中特征的取值为 i 的那些数据样本的集合（删掉被选中特征那一
列）
        if 被选中特征确实有 i 这个取值
            将 tem 与父节点连接起来
        else 释放 tem 节点
    for i=0:子节点数目;
        recursive(p->children[i]);
}
void validate(int index,Node *p){
    验证集使用决策树验证 :
    if 到达叶子节点 or 当前节点被剪枝
        取得其标签作为该验证集样例的预测结果存储进数组 vali_label_result[]中;
        return ;
    for i=0:children.size():
        找到了对应当前验证集文本特征值的节点,
        则进入这个子节点继续往下遍历
        validate(index,p->children[i]);
    if 上面没有找到对应的特征值子节点
        说明验证集出现训练集没有的新情况
        多少表决去标签存放数组 vali_label_result[]中;
}
void prune(Node* p){
    if 到达叶节点 return;
    for i=0:children.size():
        prune(p->children[i]);//后序遍历
```

```
        p->prune_or_not=1;
        decide_final_label(p);
        调用 validate()函数，验证集使用剪枝后的决策树预测
        根据数组 vali_label_result[]和正确答案数组 valilabel[]对比计算准确率
        if 准确率没有降低
            剪枝
        else p->prune_or_not=0;//不剪枝
}
void initial(){
    Readtext();
    初始化根节点：初始化数据集、特征集等等参数
}
void Readtext(){
    读取 train.csv 文件，每五个样本取前四个进训练集数组，
    取第五个进验证集数组。且分别用一维数组记录其标签
}
void test_recursive(int index,Node* p){
    用决策树预测测试集标签；
    与 validate 函数类似，只是预测结果存放数组不同，不作赘述；
}
void output_test_result(){文件流输出测试集预测结果}
void Readtest(){读取测试集文本进测试集数组}
int main(){
    initial(); //初始化根节点
    recursive(root);//开始递归建立决策树
    for(int i=0;i<valicnt;i++)//未剪枝前，验证集使用决策树预测
        validate(i,root);
    计算验证集预测结果准确率
    prune(root);//剪枝
    Readtest();
    for(int i=0;i<testcnt;i++)//决策树预测测试集
        test_recursive(i,root);
    output_test_result();//输出结果到 txt 文件
}
```

3、关键代码截图（带注释）

①ID3 方法代码：包含三个函数 ID3() 、Conditional_entropy() 和 Empirical_entropy()

```
double Empirical_entropy(Node *p)//求当前数据集的经验熵 HD
{
    double HD;
    double pos=0,neg=0;
    for(int i=0;i<p->datasize;i++)//统计数据集中正标签和负标签的数目
    {
        if(p->Label[i]==1) pos++;
        else neg++;
    }
    HD=-1*(pos/p->datasize)*log(pos/p->datasize)-((neg/p->datasize)*log(neg/p->datasize));
    return HD;
}
double Conditional_entropy(int attr_index,Node *p)//求索引位置为attr_index的特征的条件熵HDA
{
    int maxnum=0,minnum=1000;
    for(int i=0;i<p->datasize;i++){//先遍历找出当前特征的特征值的最小值和最大值
        if(p->Data[i][attr_index]<minnum) minnum=p->Data[i][attr_index];
        if(p->Data[i][attr_index]>maxnum) maxnum=p->Data[i][attr_index];
    }
    double HDA=0;
    for(int i=minnum;i<=maxnum;i++){//遍历所有可能的取值
        double pos=0,neg=0,cnt=0;
        for(int j=0;j<p->datasize;j++){//遍历所有样本
            if(p->Data[j][attr_index]==i){//找出取值为i 的所有样本
                cnt++;//记录取值为i 的样本的数量
                if(p->Label[j]==1) pos++;
                else neg++;
            }
        }
        double A=0,B=0,C=0;
        if(cnt!=0){//cnt==0即没有这个特征值时不计算，cnt!=0即存在这个特征值时计算
            A=(cnt/p->datasize)*-1;
            if(pos!=0) B=pos/cnt*log(pos/cnt);
            if(neg!=0) C=neg/cnt*log(neg/cnt);
            HDA+=A*( B + C );   //累加
        }
    }
    return HDA;
}
int ID3(Node *p)//用ID3从当前节点的特征集当中选择一个特征来分裂
{
    double HD=Empirical_entropy(p);//求当前数据集的经验熵
    double HDA[10];
    for(int i=0;i<p->attrsize;i++)
        HDA[i]=Conditional_entropy(i,p);//求第i个特征，在当前数据集的经验熵
    double G[10],maxG=-100;
    int max_index=0;//信息增益最大的特征的索引位置
    for(int i=0;i<p->attrsize;i++)//遍历所有特征，得到他们的信息增益
    {
        G[i]=HD-HDA[i];
        if(G[i]>maxG){  maxG=G[i];  max_index=i;}
    }
    return p->Attr[max_index];//最终返回的是这个特征而不是其索引位置
}
```

②C4.5方法：包含函数SplitInfo()和C4_5()，还调用了上面的函数Conditional_entropy() 和Empirical_entropy()

```
double SplitInfo(int attr_index,Node* p)//计算索引位置为attr_index的特征的熵
{
    int maxnum=0,minnum=1000;
    for(int i=0;i<p->datasize;i++)//先遍历找出当前特征的特征值的最小值和最大值
    {
        if(p->Data[i][attr_index]<minnum) minnum=p->Data[i][attr_index];
        if(p->Data[i][attr_index]>maxnum) maxnum=p->Data[i][attr_index];
    }
    double sp_info=0;//当前特征的熵
    for(int i=minnum;i<=maxnum;i++)//遍历所有可能的取值
    {
        double cnt=0;//取值为i的样本的数量
        for(int j=0;j<p->datasize;j++)//遍历所有样本
        {
            if(p->Data[j][attr_index]==i)//找出取值为i的所有样本
                cnt++;//记录取值为i的样本的数量
        }
        if(cnt!=0)  sp_info+=-1*(cnt/p->datasize)*log(cnt/p->datasize);
    }
    return sp_info;
}
```

```
int C4_5(Node* p)//用C4.5方法从当前节点的特征集当中选择一个特征来分裂
{
    double gainRatio[10];//信息增益率
    double HD=Empirical_entropy(p);//计算当前数据集的经验熵
    for(int i=0;i<p->attrsize;i++)//对当前所有特征求信息增益率
    {
        double HDA=Conditional_entropy(i,p);//条件熵
        double sp_info=SplitInfo(i,p);//特征的熵
        gainRatio[i]=(HD-HDA)/sp_info;//信息增益率
    }
    double maxgR=0;
    int max_index=0;//信息增益率最大的特征的索引位置
    for(int i=0;i<p->attrsize;i++)//求信息增益率最大的特征的索引位置
    {
        if(gainRatio[i]>maxgR)
        {
            maxgR=gainRatio[i]; max_index=i;
        }
    }
    return p->Attr[max_index];//最终返回的是这个特征而不是其索引位置
}
```

③CART 方法：包含函数 gini()和 CART()：

```
double gini(int attr_index,Node* p) //计算索引位置为attr_index的特征的gini系数
{
    int maxnum=0,minnum=1000;
    for(int i=0;i<p->datasize;i++){//先遍历找出当前特征的特征值的最小值和最大值
        if(p->Data[i][attr_index]<minnum) minnum=p->Data[i][attr_index];
        if(p->Data[i][attr_index]>maxnum) maxnum=p->Data[i][attr_index];
    }
    double gini=0;
    for(int i=minnum;i<=maxnum;i++){//遍历所有可能的取值
        double pos=0,neg=0,cnt=0;
        for(int j=0;j<p->datasize;j++){//遍历所有样本
            if(p->Data[j][attr_index]==i)//找出取值为i的所有样本
            {
                cnt++;//记录取值为i的样本的数量
                if(p->Label[j]==1) pos++;
                else neg++;
            }
        }
        double A=0,B=0,C=0;
        if(cnt!=0){//如果这个特征取值i存在，累加进基尼系数中
            A=(cnt/p->datasize);
            B=(pos/cnt)*(pos/cnt);
            C=(neg/cnt)*(neg/cnt);
            gini+=A*(1- B - C );
        }
    }
    return gini;//返回这个特征的基尼系数
}
```

```
int CART(Node* p)//用CART方法选取特征
{
    double Gini[10];
    for(int i=0;i<p->attrsize;i++)//计算得到每个特征的基尼系数
        Gini[i]=gini(i,p);
    double minnum=1000;  int min_index=0;//基尼系数最小的特征的索引位置
    for(int i=0;i<p->attrsize;i++)//得到基尼系数最小的特征的索引位置
    {
        if(Gini[i]<minnum)
        {
            minnum=Gini[i];min_index=i;
        }
    }
    return p->Attr[min_index];//最终返回的是这个特征而不是其索引位置
}
```

④边界条件函数

```
bool meet_with_bound(Node* p)//每次递归时调用，用以判断是否满足边界条件
{
    //边界条件一：所有数据集样本的标签都相同
    bool flag1=1;
    int firstone=p->Label[0];//以第一个标签为准，看后面的是否都跟他相同
    for(int i=0;i<p->datasize;i++){
        if(p->Label[i]!=firstone){
            flag1=0; break;
        }
    }
    if(flag1==1) return 1;
```

```
    //边界条件二：所有数据集的每个特征的取值都相同，这样会导致
    //每次取任意一个特征都只能生成同一个节点一直到最后特征取完。
    //但在这次实验中，训练集中并没有这种情况出现
    bool flag2=1;
    for(int j=0;j<p->attrsize;j++){
        int maxnum=0,minnum=1000;
        for(int i=0;i<p->datasize;i++){
            if(p->Data[i][j]<minnum) minnum=p->Data[i][j];
            if(p->Data[i][j]>maxnum) maxnum=p->Data[i][j];
        }
        if(maxnum!=minnum){ //每个特征，都判断其取值的最大值和最小值是否相等，相等即取值相等。
            flag2=0; break;
        }
    }
    if(flag2==1) return 1;
    //如果都不符合以上边界条件，则说明没有到达边界，继续分裂
    return 0;
}
```

⑤递归函数 recursive()：

```
232 void recursive(Node *p)//递归生成全部节点
233 {
234     if(meet_with_bound(p))//如果判断到达边界，则当做叶子节点处理，然后返回
235     {
236         cnt_of_leave++;//叶子节点数+1
237         decide_final_label(p);//决定这个叶子结点的标签
238         return;
239     }
240     //不是叶子节点，则继续分裂/分支
241     int attr_chosen=choose_attr(p);//选择一个当前特征集中最优代表性的特征
242     p->attr_chosen=attr_chosen;//记录下当前节点选择了哪个特征进行分裂
243     int index_chosen;//寻找出这个被选中的特征在当前节点的特征集中的索引位置
244     for(int i=0;i<p->attrsize;i++)if(p->Attr[i]==attr_chosen)
245     {
246         index_chosen=i;break;
247     }
248     int maxnum=0,minnum=1000;
249     for(int i=0;i<p->datasize;i++)//先遍历找出被选中特征的特征取值的最小值和最大值
250     {
251         if(p->Data[i][index_chosen]<minnum) minnum=p->Data[i][index_chosen];
252         if(p->Data[i][index_chosen]>maxnum) maxnum=p->Data[i][index_chosen];
253     }
```

```
254     for(int i=minnum;i<=maxnum;i++)//遍历该特征所有可能的取值
255     {
256         Node *tem=new Node;
257         //新建特征集，在父节点特征集的基础上删掉被选中的特征
258         tem->datasize=0;    tem->attrsize=0;    tem->attr_num=i;//记录当前新建节点的特征取值
259         tem->prune_or_not=0;//默认不剪枝
260         int temcnt=0;//记录该新节点的特征数量，其实直接p->attrsize-1也可以，不过这样更容易检查调试
261         for(int j=0;j<p->attrsize;j++)if(j!=index_chosen)
262         {
263             tem->Attr[temcnt]=p->Attr[j];
264             temcnt++;
265         }
266         tem->attrsize=temcnt;
```

```
        tem->attrsize=temcnt;
        //新建数据集，取属性值为i的数据
        int temcnt1=0;//记录特征取值为i的文本数目
        for(int j=0;j<p->datasize;j++)//遍历数据集
        {
            if(p->Data[j][index_chosen]==i)//选中这些特征取值为i的行
            {
                tem->Label[temcnt1]=p->Label[j];
                int temcnt2=0;
                for(int k=0;k<p->attrsize;k++)//复制这一整行
                {
                    if(k!=index_chosen)//文本中不是被选中特征的那一列，都复制
                    {
                        tem->Data[temcnt1][temcnt2]=p->Data[j][k];
                        temcnt2++;
                    }
                }
                temcnt1++;
            }
        }
        tem->datasize=temcnt1;
        //上面是新建子结点，我不管他数据集里有没有i这个属性值，直接就建了，
        //建完再判断temcnt是否为0，不为0才与父节点连接起来。
        if(temcnt1!=0)
            p->children.push_back(tem);
        else
            free(tem);//没用的节点释放掉，节省空间
    }
    for(int i=0;i<p->children.size();i++)
        recursive(p->children[i]);
}
```

⑥验证集使用决策树预测结果，validate()函数

```
void validate(int index,Node *p)//验证集使用决策树验证
{
    if(p->children.size()==0||p->prune_or_not==1)//到达叶节点或者在当前节点剪枝（即将当前节点变成叶子节点）
    {
        vali_label_result[index]=p->final_label;//取得其标签作为该验证集样例的预测结果
        cnt_of_arriving_leaf++;
        return ;
    }
    bool flag=0;
    for(int i=0;i<p->children.size();i++){
        if(p->children[i]->attr_num==vali[index][p->attr_chosen])//找到了对应特征值的节点，则进入这个子节点继续往下遍历
        {
            validate(index,p->children[i]);
            flag=1;
        }
    }
    if(flag==0)//如果当前节点有children，却没有对应的值的children，只能说验证集出现了训练集里没有的情况，这时候多数投票
    {
        cnt_of_not_arriving_leaf++;//没有到达叶子结点，中途就停下了
        int pos=0,neg=0;
        for(int i=0;i<p->datasize;i++)
        {
            if(p->Label[i]==1) pos++;
            else neg++;
        }
        if(pos>=neg) vali_label_result[index]=1;
        else vali_label_result[index]=-1;
    }
}
```

⑦main 函数跟伪代码中所写差不多，还有其他简单的函数比如 initial()、decide_final_label（多数投票）等、还有已经实现过的函数 Readtext()读取文本等不作展示。

3. 创新点&优化（如果有）

优化：后剪枝提升决策树的泛化性能

后剪枝的剪枝过程：后序遍历决策树，尝试用叶子节点代替子树，这个叶子节点所标识的类别通过多数投票确定，如果剪去这颗子树后验证集准确率没有下降，则用叶子结点替代此节点，如果准确率下降，则恢复原状。

代码：prune()函数，直接 prune(root);调用即可，accuracy 是完全决策树的准确率：

```
void prune(Node* p)//利用验证集对决策树进行剪枝
{
    if(p->children.size()==0) return;//到达叶节点返回
    for(int i=0;i<p->children.size();i++)//后序遍历
    {
        prune(p->children[i]);
        p->prune_or_not=1;//尝试剪枝
        decide_final_label(p);//当前节点当做叶子结点，则需要设置标签，多数投票取标签
        //用验证集算准确率
        for(int i=0;i<valicnt;i++) validate(i,root);
        double right=0;
        for(int i=0;i<valicnt;i++)
        {
            if(vali_label_result[i]==valilabel[i]) right++;
        }
        double temac=right/valicnt;
        if(temac>=accuracy)//如果剪枝后准确率没有降低，则删除其所有子节点，将其变成叶子结点
        {
            p->children.clear();//子节点都删除
            accuracy=temac;
        }
        else
            p->prune_or_not=0;//恢复原状，不剪去该节点
    }
}
```

剪枝效果：

```
C:\Users\Administrator\Desktop\人工智能\lab4_Decision_Tree\
textcnt=525  valicnt262
cnt_of_leave344
cnt_of_arriving_leaf=215
cnt_of_not_arriving_leaf=47
right=169
accuracy=0.645038
cnt_of_pruned_leaves:41
After pruning:accuracy=0.683206
testcnt=300
test_cnt_of_arriving_leaf=280
test_cnt_of_not_arriving_leaf=20
testpos=137
testneg=163

--------------------------------
Process exited after 1.032 seconds with return v
请按任意键继续. . .
```

ID3 方法，可见剪枝前验证集准确率为 0.645，剪枝后准确率上升到 0.683，且剪去了 41 个子树，将其换成了叶子节点。

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）

从网上找到一个简单的数据集

日志密度	好友密度	是否使用真实头像	账号是否真实
s	s	no	no
s	l	yes	yes
l	m	yes	yes
m	m	yes	yes
l	m	yes	yes
m	l	no	yes
m	s	no	no
l	m	no	yes
m	s	no	yes
s	s	yes	no

转化为数据集，其中 s->1,m->2,l->3；no->0,yes->1。

```
1,1,0,0
1,3,1,1
3,2,1,1
2,2,1,1
3,2,1,1
2,3,0,1
2,1,0,0
3,2,0,1
2,1,0,1
1,1,1,0
```

每三个样本，取前两个入训练集，第三个入验证集，代码运行结果：

```
C:\Users\Administrator\Desktop\人工智能\lab4_Decis
Train set:
1 1 0
1 3 1
2 2 1
3 2 1
2 1 0
3 2 0
1 1 1
Vali set:
3 2 1
2 3 0
2 1 0
```

用 ID3 方法选择特征：

HD= -(4/7)*ln(4/7)-(3/7)*ln3/7)=0.682908

H(D|A=0)=3/7*[- 1/3*ln(1/3) - 2/3*ln(2/3)] + 2/7*[-1/2*ln(1/2) -1/2*ln(1/2)]

+ 2/7*[-1*ln(1) -0]=0.272791+0.198042+0=0.470834;

H(D|A=1)=2/7*[-1*ln(1) -0] + 3/7*[-1*ln(1)-0] + 2/7*[-1*ln(1)-0]=0

H(D|A=2)=3/7*[-1/3*ln(1/3)-2/3*ln(2/3)]+4/7*[-3/4*ln(3/4)-1/4*ln(1/4)]=0.594126;

G=HD-HDA 很容易算，与代码结果相符

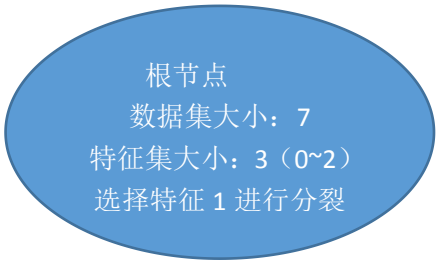
```
HD=0.682908
HDA0=0.470834
HDA1=0
HDA2=0.594126
G0=0.212074
G1=0.682908
G2=0.0887819
```

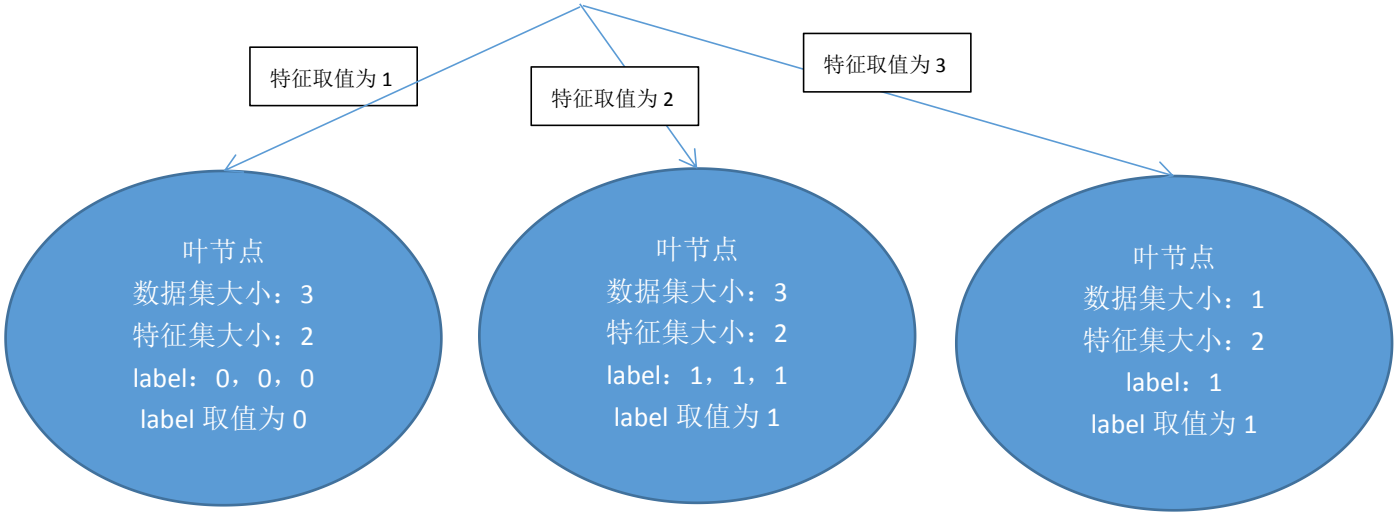
其中特征 1 的信息增益最大，因此选择特征 1 进行分裂

```
当前节点datasize=7
当前节点attrsize=3
选择特征：1
最大特征值:3  最小特征值:1
第0个子节点    特征集大小:2    数据集大小:3
第1个子节点    特征集大小:2    数据集大小:3
第2个子节点    特征集大小:2    数据集大小:1
有一个叶子节点
当前节点datasize=3
0 0 0
有一个叶子节点
当前节点datasize=3
1 1 1
有一个叶子节点
当前节点datasize=1
1
cnt_of_leave3
cnt_of_arriving_leaf=3
cnt_of_not_arriving_leaf=0
right=2
准确率0.666667

--------------------------------
```

转化作树结构表示：





验证集使用决策树预测时，

第一个样例 3　2　1，特征 1 的取值为 2，进入第二个子节点，预测标签为 1，预测正确；

第二个样例 2 3 0，特征 1 取值为 3，进入第三个子节点，预测标签为 1，预测错误；

第三个样例 2 1 0，特征 2 取值为 1，进入第一个子节点，预测标签为 0，预测正确

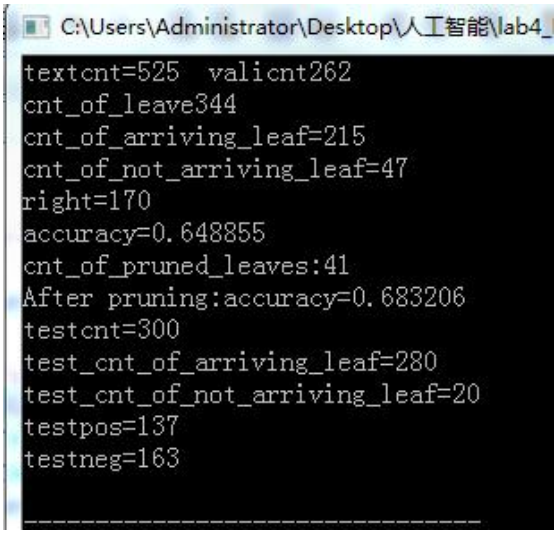
因此准确率为 0.66667

2. 评测指标展示及分析（如果实验题目有特殊要求，否则使用准确率）

读取 train.csv 时，每三个样本，取前两个入训练集，第三个入验证集

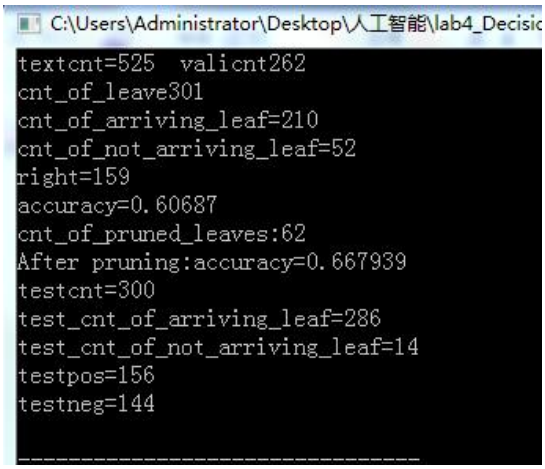
ID3 方法

准确率 0.648855，剪枝后准确率 0.683206



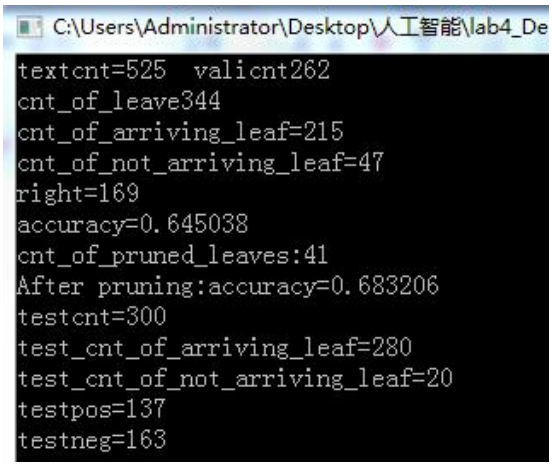
C4.5 方法

准确率 0.60687，剪枝后准确率 0.667939



CART 方法

准确率 0.645038，剪枝后准确率 0.683206



三、 思考题

- **决策树有哪些避免过拟合的方法？**

过拟合的概念：决策树过拟合在外形上表现为树的结构相对较复杂。一棵过拟合的树在泛化能力的表现上很差，也就是说这样的树对于训练集的准确率很高但是对于测试集或者未来的数据表现很差。

（1）剪枝。完全的决策树会导致过度拟合，而使用一个验证集来纠正决策树，即在不影响决策树准确率的情况下剪去多余的节点。对于完全决策树中的每一个非叶子节点的子树，都尝试把它替换成叶子节点，该叶子节点的标签用子树所覆盖训练样本中存在最多的那个类来代替（即多数表决），这样就产生了一个简化决策树，然后比较这两个决策树在测试数据集中的准确率，如果简化决策树准确率没有降低，那么该子树就可以替换成叶子节点，这样就实现了决策树的泛化。

（2）数据集扩增。我们在使用训练数据训练模型，通过这个模型对将来的数据进行拟合，而训练数据与将来的数据是独立同分布的。过拟合的的问题在于将来的数据会出现训练数据中没有出现过的样例，导致判断错误，那么如果能得到更多的训练数据，则往往估计与模拟

地更准确。

（3）正则化方法。正则化为了降低模型的复杂度，对模型向量进行“惩罚”，从而避免单纯最小二乘问题的过拟合问题。由于正则化公式复杂且繁多，只了解大概的原理：规则化项的引入，在训练（最小化 cost）的过程中，当某一维的特征所对应的权重过大时，而此时模型的预测和真实数据之间距离很小，通过规则化项就可以使整体的 cost 取较大的值，从而，在训练的过程中避免了去选择那些某一维（或几维）特征的权重过大的情况，即过分依赖某一维（或几维）的特征。也就是说采用正则化方法会自动削弱不重要的特征变量，自动从许多的特征变量中”提取“重要的特征变量，减小特征变量的数量级，从而避免过度拟合。

- **C4.5 相比于 ID3 的优点是什么？**

（1）ID3 算法在选择特征进行节点分支时，采用信息增益作为评价标准。信息增益的缺点是倾向于选择取值较多的属性，在有些情况下这类属性可能不会提供太多有价值的信息。而 C4.5 采用信息增益率作为评价标准，克服了用信息增益来选择属性时偏向选择值多的属性的不足，且其分类规则易于理解，准确率较高。

（2）ID3 算法只能对描述属性为离散型属性的数据集构造决策树。而 C4.5 既可以处理离散型描述属性，也可以处理连续性描述属性。

- **如何用决策树来判断特征的重要性？**

首先按正常步骤用原来的训练集建立决策树，并且用验证集验证得到准确率。然后对这个训练集的某一个特征的列的数据进行随机加噪，再用来构建决策树，再用相同的验证集验证得到准确率，如果这个准确率对比之前的准确率有大幅度的下降，则说明了这个特征十分重要，如果准确率变化不大，说明这个特征影响力不大。

|---------- 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