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	15M2	专业（方向）	互联网
学号	15352218	姓名	林燕娜

一、 实验题目

决策树算法

二、 实验内容

1. 算法原理

决策树：

决策树是机器监督学习中非常常见的算法，可以用于分类和回归，本次实验主要是用于二分类，结果只有 1 和-1.

决策树的建立过程，就是选择每一个节点的特征，然后按这种节点的分支值划分例子再选出每个分支节点的特征，直到整个树建立完成。其递归的终止条件有 2 个：

-> 当前划分的例子中的结果一致，都是 1 或者-1；

-> 划分到目前为止，已经考虑了所有的属性，则判定结果为划分数据中较多的那个。

-> 数据集的剩余的属性的取值完全一致，则判定结果为划分数据中较多的那个。

决策树算法是一个贪心算法的过程，每一步都选出当前属性中效果最好的属性。而被选出的这个属性，在其之后的子孙结点中，就不会再次出现。

所以，到达每一个叶子节点，所经过的路程的属性都是唯一的，不会出现重复的情况。

关于选择树节点属性的三种算法：

- **ID3：**

ID3,就是求信息增益，其公式是：$g(D,A)=H(D)-H(D|A)$。

其中，H(D)是数据集 D 的经验熵，H(D|A)是在属性 A 下 D 的条件熵。一般情况下，可以将熵理解为不确定性。H(D)是原本数据集的不确定性，而 H(D|A)是在属性 A 下 D 的不确定性。我们是希望选出不确定性越小的属性，使最后的结果的确定性大一点。所以选择 H(D|A)小一点的属性 A。

而增益的公式是 $g(D,A)=H(D)-H(D|A)$，所以可以理解为增益是在属性 A 的情况下，数据集的不确定性改变了多少。改变的越多，增益越大，说明 H(D|A)越小。所以，我们选出增益最大的属性作为决策点。

- **C4.5**

C4.5 又叫信息增益率，是在 ID3 增益的基础上计算增益率。

引入 C4.5 的原因是因为 ID3 信息增益度量存在一个明显的缺点，ID3 会偏向选取分支较多的属性，因为分支较多的属性一般情况下增益会比较大。为了弥补这个缺点，C4.5 就引入了一个分裂信息来惩罚分支多的属性。

分裂信息，又叫数据集 D 关于 A 的熵 SplitInfo(D,A)。计算公式如下：

$$SplitInfo(D,A)=H(A)$$

由计算公式可以看出，分裂信息其实也就是属性 A 自身的不确定性。分支越多的信息，其不确定性会高于分支较少的。最极端的例子就是每个分支的值都不同，这样子的不确定性就是 $-\frac{1}{n}*\log_2(\frac{1}{n})*n=\log_2 n$. 则分支数 n 越多，其不确定性越大。也就是分支越多，其 SplitInfo(D,A)也就越大。所以，将信息增益 ID3 除以其对应的属性的熵，就可以将其属性的分支数量考虑进去，不会一味偏向于分支数量多的属性。

所以信息增益率的公式：

$$gRatio(D,A)=\frac{g(D,A)}{SplitInfo(D,A)}$$

同理，信息增益率也是越大越好，我们选择信息增益率最大的属性作为决策点。

- **CART**

CART 算法引入了一个参数，就是 gini 系数，gini 系数的计算公式如下：

$$gini(D,A)=\sum_{j=1}^{v}p(A_j)*(1-\sum_{i=1}^{n}p_i^{\,2})$$

因为在本次试验中最终的判定结果只有两个，要么是 1，要么是-1。所以 n = 2，并且当一个概率为 P 的时候，另一个概率为 1-P。上述公式进一步变为：

$$gini(D,A)=\sum_{j=1}^{v}p(A_j)*(1-p_i^{\,2}-(1-p_i)^2)$$

复杂性代价剪枝：

因为决策树的数据集本身存在的问题，会造成过拟合，而解决过拟合可以用剪枝的方法，其中一种就是复杂性代价剪枝。

复杂性代价剪枝的公式是：

$e_c=\frac{\sum_{l=1}^{L}(E_l+\alpha)}{\sum_{l=1}^{L}N_l}$ 其中，是子树的叶子结点，α是每个叶子节点的惩罚。E_l 是第 l 个叶子节点中错误的数据个数，N_l 是第 l 个叶子节点的所有数据个数。

复杂性代价剪枝，就是对一颗子树，考虑其所有的叶子节点的错误综合 e_c 与假设只有

一个根节点，根结点判定为所有的叶子节点的数据中的众数。计算其错误：$e_c' = \frac{E+\alpha}{\sum_{l=1}^{L} N_l}$。

比较叶子节点的错误e_c和只有一个根结点的错误e_c'的大小。如果$e_c \geq e_c'$。则说明一个根结点的错误小于多个叶子的错误，也就没必要存在那么多个叶子，就进行剪枝。

而对于惩罚系数的理解：

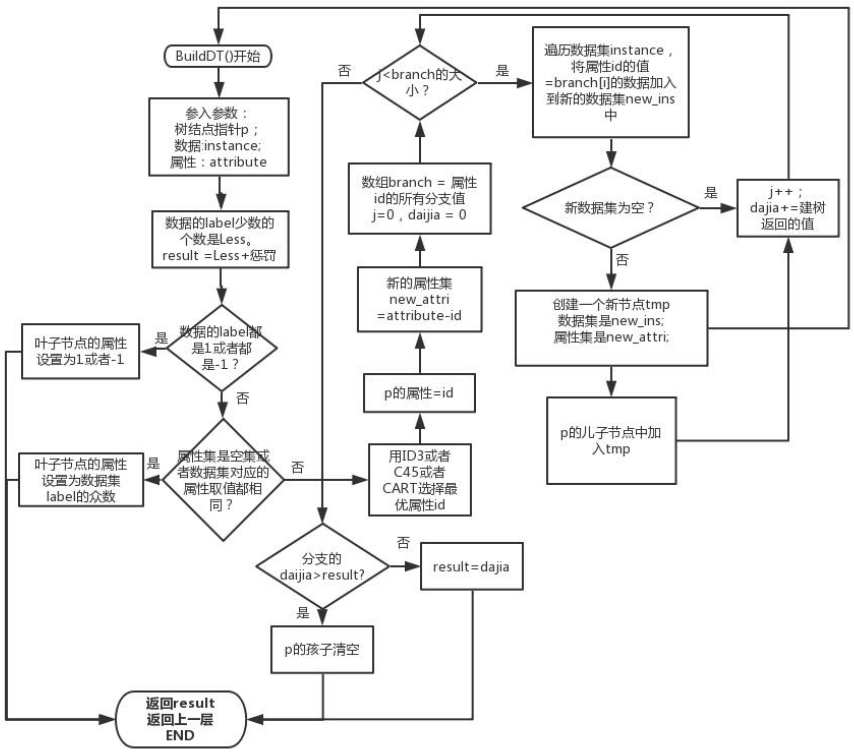
如果不需要剪枝，也就是需要满足：

$$e_c \leq e_c' \Rightarrow \sum_{l=1}^{L}(E_l+\alpha) \leq E+\alpha \Rightarrow (L-1)\alpha \leq E - \sum_{l=1}^{L} E_l$$

而$E - \sum_{l=1}^{L} E_l$就是分支比不分支要多的正确个数。所以，只有当分支多增加的正确个数不小于（叶子个数-1）*惩罚系数，才可以考虑分支。

2. 伪代码/流程图

➢ 递归建树剪枝流程图：



3. 关键代码（带注释）

➢ 递归建树：

->根据递归终止的 3 个条件进行树建立的结束；

->根据所选的算法选出当前最合适的属性；

->根据选出的属性更新剩余的属性集合，并根据选出的属性的分支进行例子的划分，递归建树。

->利用建树的递归特点，顺带完成剪枝。

```
double BuildDecisionTree(node* current_node,vector <vector<int> > remain_instance, vector <int> remain_attribute,int way){
    int less_number = count_most(remain_instance);
    if(current_node == NULL) current_node = new node();
        //递归终止条件
    if(All_is_same(remain_instance)==1){
        current_node->attribute = 1;
        return less_number+PUNISHMENT;
    }
    if(All_is_same(remain_instance)==-1){
        current_node->attribute = -1;
        return less_number+PUNISHMENT;
    }
    if(!remain_attribute.size()||All_attri_same(remain_instance,remain_attribute)){
        current_node -> attribute = Most_Label(remain_instance);
        return less_number+PUNISHMENT;
    }

    //根据相应的算出对应的父节点的属性
    int choose_attribute;
    if(way == 2)    //GIni
        choose_attribute = Gini(remain_instance,remain_attribute);
    if(way == 1)    //C45
        choose_attribute = C4_5_or_ID3(remain_instance,remain_attribute,true);
    if(way == 0) //ID3
        choose_attribute = C4_5_or_ID3(remain_instance,remain_attribute,false);

    current_node -> attribute = choose_attribute;
    //更新剩余的属性
    vector <int> new_attribute;
    vector <vector <int> > new_instance;

    for(int i = 0;i<remain_attribute.size();i++){
         if(remain_attribute[i]!=choose_attribute) new_attribute.push_back(remain_attribute[i]);
    }
    //根据父亲节点的属性的分支，进行例子的分割,每个分支递归建树
    vector<int> branch = branches[choose_attribute];

```

```
    //sum_of_childs_false 记录所有的分支的错误
    double sum_of_childs_false = 0;
    for(int j = 0;j<branch.size();j++){
        //对数据进行分割
        for(int i = 0;i<remain_instance.size();i++){
            if(remain_instance[i][choose_attribute]==branch[j]){
                new_instance.push_back(remain_instance[i]);
            }
        }
        //某个分支没有了对应的例子，不处理  假设为没这个分支
        if(new_instance.size() == 0){
            continue;
        }
        else{
            node* new_node = new node();
            new_node -> father_attribute = current_node->attribute;
            new_node -> arrive_value = branch[j];
             sum_of_childs_false += BuildDecisionTree(new_node,new_instance,new_attribute,level+1,current_node->attribute,way);
            current_node -> childs.push_back(new_node);
            new_instance.erase(new_instance.begin(),new_instance.end());
        }
    }
    //分支后错误率高，则剪枝
    if(sum_of_childs_false>=less_number+PUNISHMENT){
        current_node->childs.erase(current_node->childs.begin(),current_node->childs.end());
        sum_of_childs_false = less_number+PUNISHMENT;
        current_node->attribute = current_node -> most_label;
    }
    return sum_of_childs_false;
}
```

4. 创新点&优化（如果有）

➢ 建树过程完成了剪枝：

利用建树的递归特点，顺带完成剪枝。最开始递归终止的肯定是叶子结点。并且整棵树的建立是有层次的结束建树的过程，其完成的顺序剪枝需要遍历的顺序大概类似，所以就顺带的在建树过程完成了剪枝。

将建树返回值设置为叶子节点的错误（包括了惩罚值）或者是某颗子树的错误。在某个节点的所有 childs 都建树完成后，将其所有的 childs 错误相加，与该节点的错误相比较（该节点的错误就是作为一个根结点，用多数投票的原则判定，则错误就是：少数数据的个数+一个叶子的惩罚）。如果所有的 childs 的错误相加>=该节点的错误，则代表该节点不分支的错误小于等于分支的错误。所以，可以不分支，这样子就达到了剪枝的效果。

如果是所有的childs的错误相加<该节点的错误,则该节点的错误更新为所有的childs的错误相加，并把该值返回给其父亲节点，进行进一步的剪枝处理判断。

三、 实验结果及分析

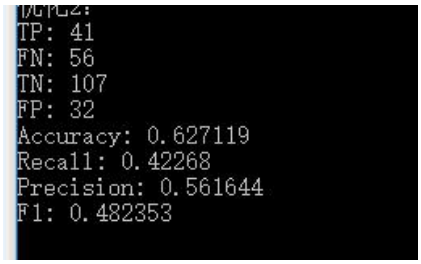
1. 实验结果展示示例（可图可表可文字，尽量可视化）

数据集的分法：

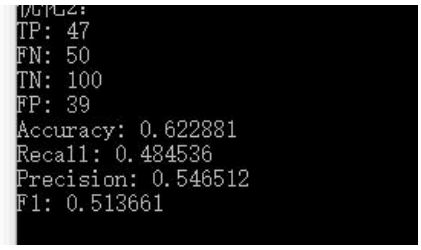
考虑到一般情况下，验证集都是训练集的20%-30%。所以选取了30%。为了+1和-1样本的尽量的公平性。在+1的样本后面抽取其总数的30%，在-1样本的后面同样抽取其总数的30%作为最后的验证集。剩余的就是训练集。

剪枝结果展示：（惩罚因子：0.7）

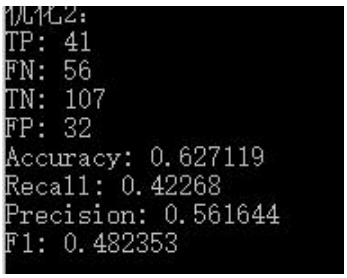
ID3:



C4.5



CART:



选择ID3算法进行验证：

训练集如下：

a	b	c	d	label
1	1	1	0	1
1	1	1	0	1
1	1	1	0	1
1	1	0	0	1
1	1	1	0	1
1	1	0	0	1
2	1	1	0	1
2	1	1	0	1
1	1	1	1	1
2	1	1	0	1
2	1	1	0	-1
2	1	1	1	-1
1	1	1	0	-1
2	1	1	0	-1
2	1	1	1	-1
1	1	0	0	-1
1	1	1	0	-1
2	1	1	1	-1
1	1	1	0	-1
2	1	1	1	-1

算出每一个属性的 ID3:

对于 a 属性：$\mathrm{H(D|A)} = -\frac{11}{20}\log 2(\frac{11}{20}) - \frac{9}{20}\log 2(\frac{9}{20}) = 0.993$，则其增益等于

$\mathrm{H}(D) - H(D|A) = 0.0667$

其他的属性也一致，具体计算可借助 excel，可得到以下表格:

	a	b	c	d	label
log2	0.520113168	1	0.137744375	0.728212946	熵H(D)
	0.413233125		0.847877164	0.180482024	
H(D|A)	0.933346293	1	0.98562154	0.90869497	1
gain	0.066653707	0	0.01437846	0.09130503	

用程序输出其计算的 gain 和选择的属性:

```
属性 0 的gain 0.0666537
属性 1 的gain 0
属性 2 的gain 0.0143785
属性 3 的gain 0.091305
选择属性  3
```

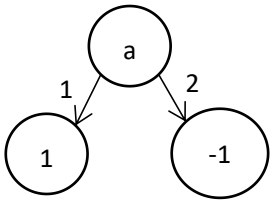
选中属性 d 后，我们按照 d 划分为两个数据集。

- 当 d 的取值等于 1 的时候

a	b	c	d	label
1	1	1	1	1
2	1	1	1	-1
2	1	1	1	-1
2	1	1	1	-1
2	1	1	1	-1

明显，最终的结果可以由 a 唯一确定，所以该选择属性 a。

所以该分支的子树为:



- 当 d 的取值等于 0 的时候：

a	b	c	d	label
1	1	1	0	1
1	1	1	0	1
1	1	1	0	1
1	1	0	0	1
1	1	1	0	1
1	1	0	0	1
2	1	1	0	1
2	1	1	0	1
2	1	1	0	1
2	1	1	0	-1
1	1	1	0	-1
2	1	1	0	-1
1	1	0	0	-1
1	1	1	0	-1
1	1	1	0	-1

- 计算除了 d 之外的 gain 增益：

	a	b	c	d	label
log2	0.647300396	0.970950594	0.183659167	#DIV/0!	熵H(D)
	0.323650198		0.783895005	#DIV/0!	
H(D|A)	0.970950594	0.970950594	0.967554172	#DIV/0!	0.970950594
gain	0	0	0.003396422	#DIV/0!	

选出增益最大的属性 c，继续划分：

- ✓ 当 c 的取值为 0 的时候：

a	b	c	d	label
1	1	0	0	1
1	1	0	0	1
1	1	0	0	-1

因为在剩下的属性 a，b 中，数据的所有值都是一样的，所以判定 label 为众数 1.

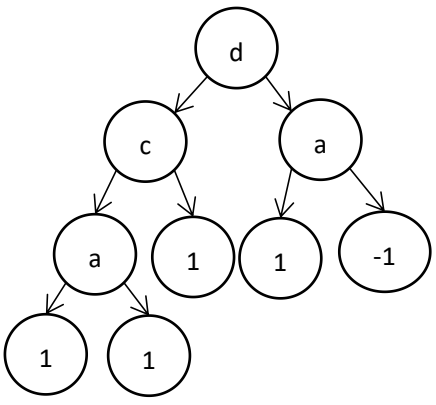
- ✓ 当 c 的取值为 1 的时候：

计算剩余的属性 a，b 的 gain。明显是 a 比较大，选择 a 属性。

	a	b	c	d	label
log2	0.574716413	0.979868757	#DIV/0!	#DIV/0!	熵H(D)
	0.404562748		#DIV/0!	#DIV/0!	
H(D|A)	0.97927916	0.979868757	#DIV/0!	#DIV/0!	0.979868757
gain	0.000589596	0	#DIV/0!	#DIV/0!	

以上面类似的方法，就可以建出最后的树。

最后的树结果如下：



而代码输出的树如下：

```
 :d
d:c   d:a
c:a   c:1   a:1   a:-1
a:1   a:1   Build Tree finished
```

其中，每一行表示每一层，：前面是其对应的父元素，而`代表的是根。

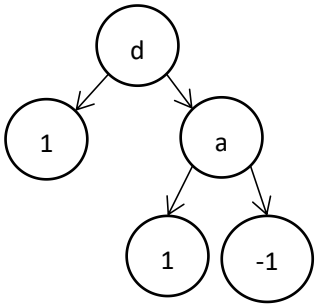
明显实验输出的图与预想的一致。建树正确。

剪枝验证：

在本次验证中，每个叶子结点的惩罚选 0.5。

- 从第一个叶子结点开始往上遍历，最左边的 a 有两个叶子节点，都判定为 1，其错误为：3+2+0.5*2 = 6，而其父节点 a 的错误为：3+2+0.5*1 = 5.5<6,所以，选择剪枝。a 替换成 1 的叶子节点，错误为 5.5.（错误已经包括了叶子的惩罚）
- 对于节点 c，情况与节点 a 一致。节点 c 的错误为：5+1+0.5*1=6.5，而叶子节点的错误总和为：5.5+1+0.5*1 = 7>6.5，所以剪枝，节点 c 替换成 1 的叶子节点，错误为 6.5。
- 对于父节点 a，其错误为：1+0.5*1 = 1.5，而其叶子节点的错误总和为：0+0+0.5*2 =1<1.5，不用剪枝。父节点 a 的错误为 1.
- 对于根节点 d，其错误为：10+0.5*1=10.5，其叶子节点的错误总和为：6.5+1=7.5<10.5，不用剪枝。

所以剪枝后的结果图如下：



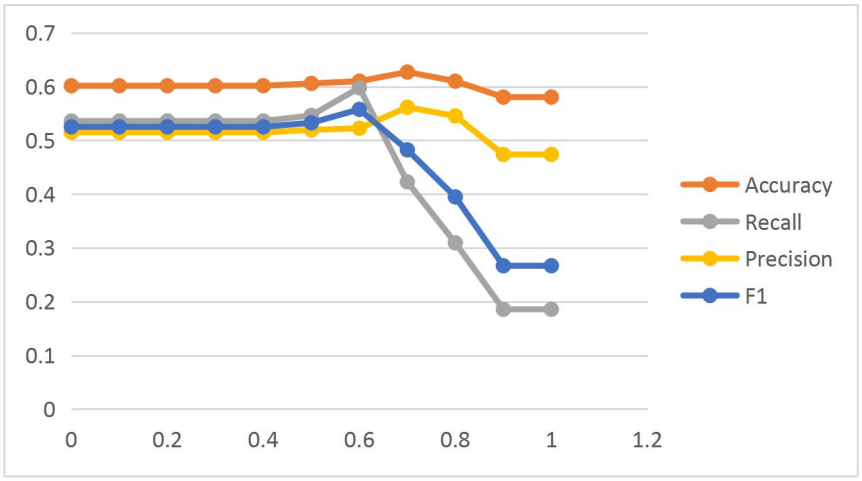
程序输出结果如下：

```
 :d
d:1   d:a
a:1   a:-1   Build Tree finished
```

明显，剪枝后结果与预期一致，剪枝成功。

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）

剪枝的惩罚系数与 Accuracy、Recall、Precision、F1 之间的关系：



总体体现较高的是在惩罚系数等于 0.6 的时候，而准确率最高在 0.7 时候，此时的 F1 只有 0.4 多。

在惩罚比较低的时候，也就是 0->0.4 区间，准确率等是几乎不变的，原因就在于不剪枝的情况下准确率会更高，也可以说是因为叶子的惩罚代价比较低，所以没有剪枝的必要。

而随着惩罚系数逐渐增大，准确率等都出现了一个上升后下降的趋势，上升是因为做了剪枝，去除了一些枝叶，使得创建的决策树不会过于拟合。而至于后面的往下降，则是因为叶子的惩罚过重，尽管有某个分支可能效果更加，但鉴于惩罚过重，没有建立分支，也就是因为惩罚过度，使得剪枝过度。如果惩罚继续增加，最极端的一个情况，就是一个分支都没有，只有一个根节点。

四、 思考题

1. 决策树有哪些避免过拟合的方法？

- **加数据**

  过拟合有可能是因为数据量太小，与现实总体相差太大。所以可以通过手机更多的数据进行训练。

- 剪枝：

  预剪枝：

  1）设定每个结点的最少的样本数量。如果某个节点处的样本小于设定的阈值，不再分支。

  2）给定一个树的最高的深度，如果超过了这个深度，就不再继续分支。

  后剪枝：

  1）错误率降低剪枝。这是最简单粗暴的方法，一旦子树的错误率可以被一个叶子结点代替，则直接把子树替换成结点。

  2）复杂性代价剪枝。在错误率的计算基础上，在分子加上一个叶子的惩罚系数，以惩罚叶子过多，结构过于复杂的情况。

- **随机森林：**

  用随机的方式建立一个森林，森林里面有很多决策树，而且每一颗决策树之

间是没有关联的。某个样本被进行预测的时候，随机森林的每一颗树都进行预测，最后以投票取众数的办法判定最终的结果。

2. C4.5 相比于 ID3 的优点是什么？

ID3 是倾向于分支数目多的属性，而 C4.5 引入 SplitInfo 算出了信息增益率。分支数目多的属性，ID3 大的同时，SplitInfo 也大，所以 C4.5 用信息增益率，消除了 ID3 的这个缺点。

ID3 只能处理离散性变量，而 C4.5 可以对连续属性进行离散化处理。

3. **如何用决策树来判断特征的重要性？**

一个属性可能在一棵树的不同层次出现，所以我们只能相对的来说，从根结点到某一个叶子节点上，越靠近根对该叶子节点的重要性就越大。