人工智能实验

实验二：ID3、C4.5、CART 决策树的实现

16 级计科教务 2 班

16337327

郑映雪

实验题目

ID3、C4.5、CART 决策树的实现

实验内容

算法原理

1、决策树

决策树是分类和回归的一种方法（本实验中用于分类）。它是一种树形结构，非叶结点为对象属性，分叉为不同的属性值。叶结点代表从根节点到叶节点所代表的这个测试对象的值。决策树易于理解，但因为要建树，实现是较为繁琐的。在结点属性选择和分类上，最重要的是要在数据集中找到一个最优的属性，对于叶节点，要从当前结点的数据集中找到一个最优值（本实验中取众数）。本实验实现的三种方法不同之处就是如何找到这个最优属性的划分。

2、ID3

ID3 是决策树的一种。在信息论中，期望信息越小，信息增益越大，信息纯度就越高。ID3 正是从信息增益来选择最优属性。

对于信息增益的计算，有如下公式：

首先计算数据集 D 本身的熵，即：

$$\mathrm{H(D)} = -\sum_{d \in D} p(d) \log p(d)$$

（其中 d 为数据集的值的集合）

对于其中一个属性，计算数据集在该属性下的条件熵，即：

$$\mathrm{H(D|A)} = \sum_{a \in A} p(a) H(D|A = a)$$

得到信息增益：

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

上文提及，信息增益越大的属性，信息纯度越高，所以对于每次准备分支之时，计算数据集中各属性的信息增益，选择信息增益最大的属性进行分类。如此递归往复，完成建树。

但是 ID3 本身评判的方式，决定了 ID3 的一个缺点——它倾向于分支比较多的属性，容易偏向有大量值的属性。所以 C4.5 模型是在 ID3 模型上做出的一个改进。

3、C4.5

C4.5 是针对 ID3 对分支较多属性的偏向性做出的一个改进。它使用了增益率来克服了这个问题。增益率顾名思义就是对于每个属性的信息增益，还要除以它本身属性的熵做作为惩罚，这样分支较多的属性，本身属性的熵也较大，就可以有效规避这个偏向性。属性熵的计算公式如下：

$$\mathrm{Splitinfo(D,A)} = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \log\left(\frac{|D_j|}{|D|}\right)$$

信息增益率的公式即为：

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

与 ID3 相同，C4.5 评判属性优劣性也是信息增益率越大越好，所以选择信息增益率最大的属性最为分支为最佳。

4、CART

CART 也是在 ID3 的基础上优化的一个决策树。在本实验中用于分类时，使用 GINI 系数作为属性选择的依据。但与之前两种树不同的是，CART 是二叉树。所以在实验

中，我在做 CART 时对数据进行了再处理，即在 CART 情况下，数据的每个属性都悲哀重新分成了两类，以保证建出的树是二叉树。

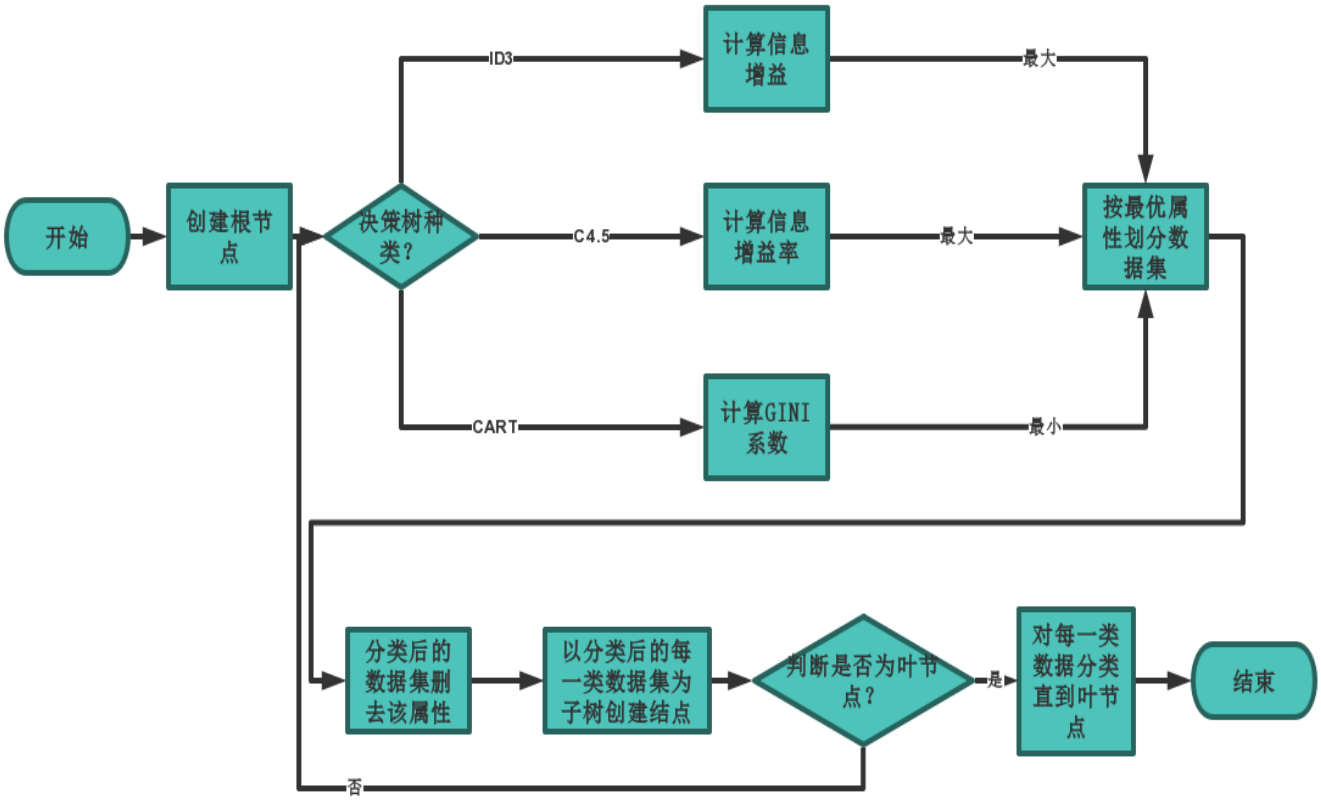
计算 GINI 系数的公式如下：

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

$$\mathrm{gini(D,A)}=\sum_{j=1}^{v}p(A_j)\times gini\left(D_j\middle|A=A_j\right)$$

由公式可以看出，GINI 系数越小，说明数据的纯度越高，所以对于 CART 树，在遍历数据的每个属性后，选择 GINI 系数最小的属性进行分类。

流程图



关键代码

1、建树准备：

设置 node 类并设定如下的特征，node 作为决策树的结点。

```
class node:
    def __init__(self, character=-1, result=-1, child={}, mostresult=-1):
        self.character = character  # 该节点分裂选择的特征
        self.result = result  # 该节点的值，若为叶节点则不为-1，默认为-1，即该节点没有值（还需继续分裂）
        self.child = child  # 字典结构，为{类别：子节点}
        self.mostresult = mostresult#该节点的数据集中多数值
```

2、计算准则

①信息增益

首先在外部计算好 H（D）并传参。在函数内部设定两个字典，分别记录各个分支的样本个数和各个分支的结果为 1 的个数，利用这两个字典中的值代入公式累加计算 H（D|A），并遍历找到最大值，此时的属性是最优属性，记录此时的属性下标并返回给建树函数。

```
def informationgain(data, hd):#计算信息增益
    datalen = len(data)
    maxgd = -10000
    prefercol = 0
    for i in range(len(data[0]) - 1):
        hda = 0
        chadic = {}  # 分支中各个情况的样本个数
        valuedic = {}  # 分支中各个情况的结果（1 或 0 的和）
        for datas in data:
            if datas[i] not in chadic.keys():
                chadic[datas[i]] = 1
                valuedic[datas[i]] = int(datas[-1])
            else:
                chadic[datas[i]] += 1
                valuedic[datas[i]] += int(datas[-1])
        for key in chadic.keys():  # 计算信息增益
            if valuedic[key] == 0 or chadic[key] == valuedic[key]:
                hda += 0
            else:
```

```
                hda += (chadic[key] / datalen) * (-valuedic[key] / chadic[key]
* log2(valuedic[key] / chadic[key]) - (1 - valuedic[key] / chadic[key]) *
log2(1 - valuedic[key] / chadic[key]))
        if hd - hda > maxgd:  # 寻找最大的信息增益
            maxgd = hd - hda
            prefercol = i
    return prefercol
```

②信息增益率

与信息增益相似，只不过加上了计算属性熵的部分，在判断时除以属性熵仍旧找到最大值即可。具体不同的地方如下所示：

```
tmpsum = 0
for values in chadic.values():
    tmpsum += values
for values in chadic.values():  # 计算属性的熵
    splitinfo += -(values / tmpsum) * log2(values / tmpsum) - (1 - values /
tmpsum) * log2(1 - values / tmpsum)
```

③GINI 系数

数据处理方式仍与另两种方法类似，只不过在代入公式和属性寻找时不同而已。

```
gini = 0
for key in chadic.keys():
    gini += (1 - (valuedic[key] / chadic[key]) ** 2 - (1 - valuedic[key] /
chadic[key]) ** 2) * (chadic[key] / tmpsum)
if gini < mingini:  # 寻找最小的 gini 系数
    mingini = gini
    prefercol = i
```

3、递归建立决策树

首先根据当前节点下的数据集，生成当前的特征字典，再将数据传输给计算准则的函数，返回一个最优属性，根据这个最优属性将数据集再次分类，并在新的数据集中删去这个属性。然后根据子节点的每一类数据都如此递归处理。此时涉及到判断的问题：当此节点是叶节点时，result 赋值，直接返回；当此节点值相同时，result 赋值，不再继续划分。

```
def createtree(root, data, hd, preresult, i, k):  # 递归建树
    characterdic = {}  # 当前的特征字典，根据当前数据集生成
    for j in range(len(data[0]) - 1):
```

```
        characterdic[j] = []
        for datas in data:
            if datas[j] not in characterdic[j]:
                characterdic[j].append(datas[j])

    if len(data) == 0:  # 若当前分类的数据集为0，则等于父节点中存在最多的值
        root.result = preresult
        return
    datadic = {}
    tmp = []
    flag = 0
    sum = 0
    for datas in data:
        sum += int(datas[-1])
    if sum < len(data) / 2:
        tmpresult = 0
    else:
        tmpresult = 1
    root.mostresult = tmpresult  # 找到当前数据集中最多出现的值
    for datas in data:
        tmp.append(datas[-1])
    if len(set(tmp)) == 1:  # 是否数据集的值相同
        flag = 1
    if len(data[0]) > 1 and flag == 0:  # 若数据集的值不相同，则建立子节点
        root.child = {}
    elif flag == 1:  # 否则，该节点的值为这个数据集的值
        root.result = int(tmp[0])
        return
    else:
        root.result = tmpresult  # 为叶节点：叶节点的值为最多出现的值
        return

    if k == '0':  # 根据K的不同选择不同的判别规则
        character = informationgain(data, hd)
    elif k == '1':
        character = informationgainprobablity(data, hd)
    else:
        character = gini(data)

    root.character = character
    for datass in characterdic[character]:
        root.child[datass] = node()
        datadic[datass] = []
```

```
    for datas in data:
        datadic[datas[character]].append(datas)  # 数据集字典为当前数据集的一个划分
    for values in datadic.values():
        for disdata in values:
            del (disdata[character])  # 在数据集中删去特征
    for key in root.child.keys():
        createtree(root.child[key], datadic[key], hd, tmpresult, 0, k)
    return
```

4、决策树输出

决策树本身的构造注定了没有一个可观的直接输出。但是 python 里的 graphviz 库可以通过建结点和边达成一个直观输出，通过对建好的树进行先序遍历从而逐步建立点和边，代码如下：

```
def printtree(root, lastname):  # 利用 graphviz 画图
    testdic = root.child
    for keys in testdic:
        if root.result != -1:
            lastname1 = root.result
            dot.node(str(lastname1))#节点名称
            dot.edge(str(lastname), str(lastname1), str(root.result))#在前两个点之间建立一条边并标记为第三个参数
            return
        else:
            lastname1 = lastname + '->' + keys#通过路径名称是独一无二的这一特性标记节点
            dot.node(str(lastname1))
            dot.edge(str(lastname), str(lastname1), keys)
            printtree(root.child[keys], lastname1)
```

5、数据判定

通过对决策树的遍历找到当前测试数据的最终测试结果。

```
    def findtheresult(testroot, testdatas):  # 遍历决策树，找到最终的值
    preresult = 0
    while testroot.result == -1:
        if testdatas[testroot.character] not in testroot.child.keys():
            return testroot.mostresult
        index = testroot.character
```

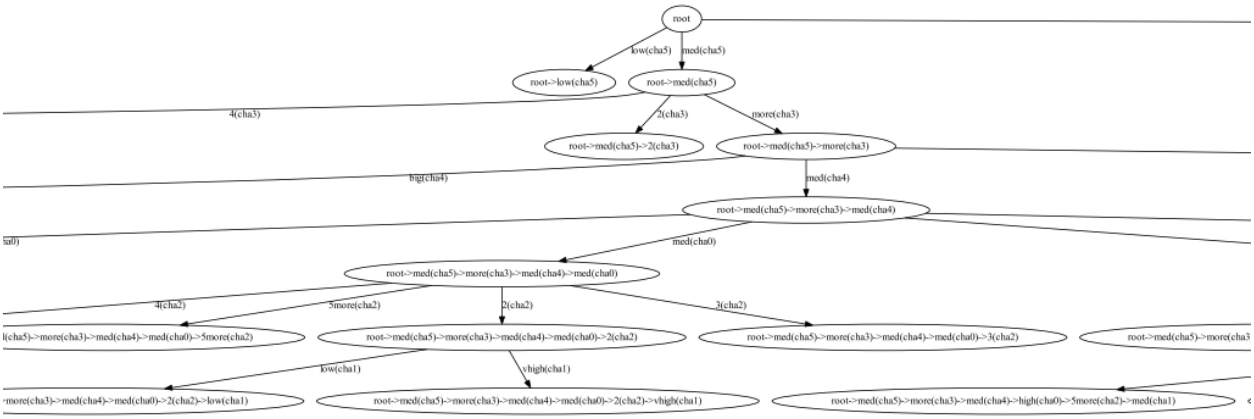
```
        testroot = testroot.child[testdatas[testroot.character]]
        del (testdatas[index])
    return testroot.result
```

实验结果及分析

实验结果展示

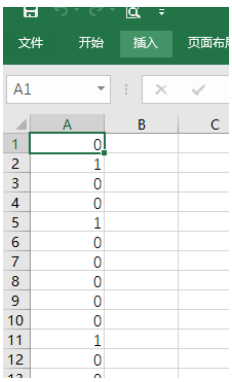
首先放上一张几乎看不清的决策树生成图片的一部分，因为它实在太宽了，没有办法通过截图展现。完整的决策树详见附件的 pdf。

注：树的结点名为从根部到这个结点的路线，每一个分类后面的（chx）的备注表示为第 x 个属性下的特征。



我对于训练集和验证集的划分尝试过 2：1 到 5:1，发现 4:1 的时候准确率最高，具体准确率会在指标中展示。而对于 GINI 系数，我一开始采取数据分类变成二叉树，但是准确率并没有多叉树高，所以依旧采取多叉树计算 GINI 系数。

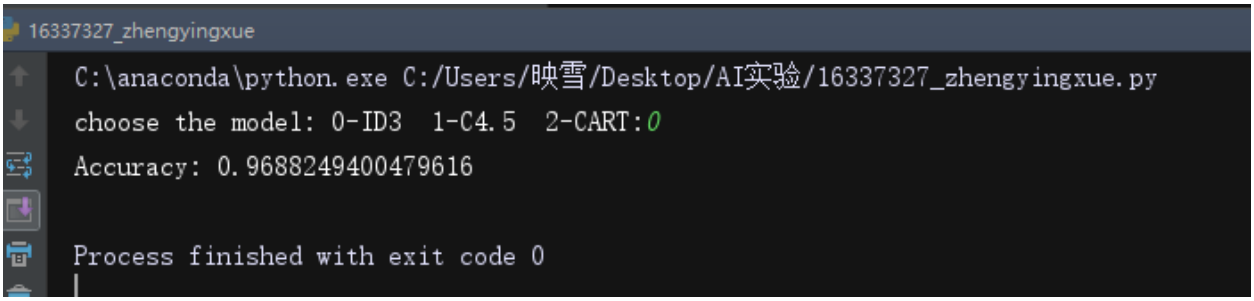
将训练集全部用来训练，测试集进行测试得出的结果如下：



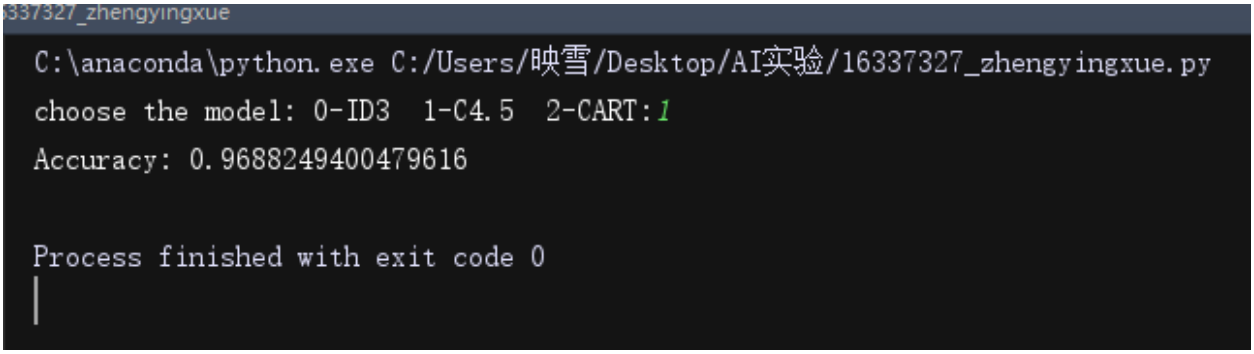
评测指标展示

对于此数据集，ID3 和 C4.5 的准确率是一样的，可见在这个数据集里，采用 ID3 和 C4.5 对于每一层的最优属性选择没有区别。也许数据集分支更多的情况下区别会比较明显。

ID3 准确率：

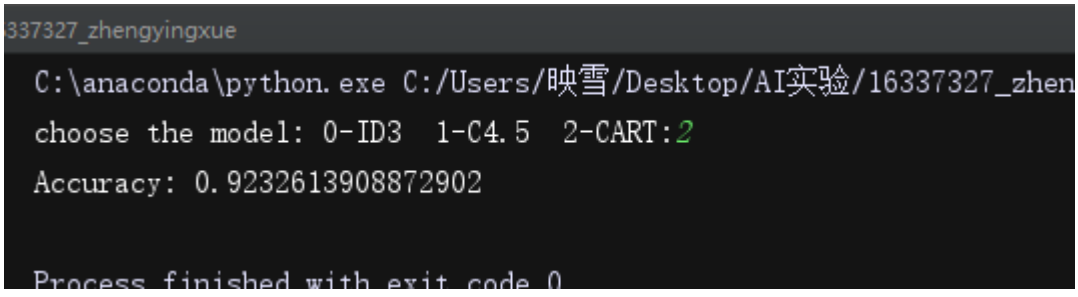


C4.5 准确率：

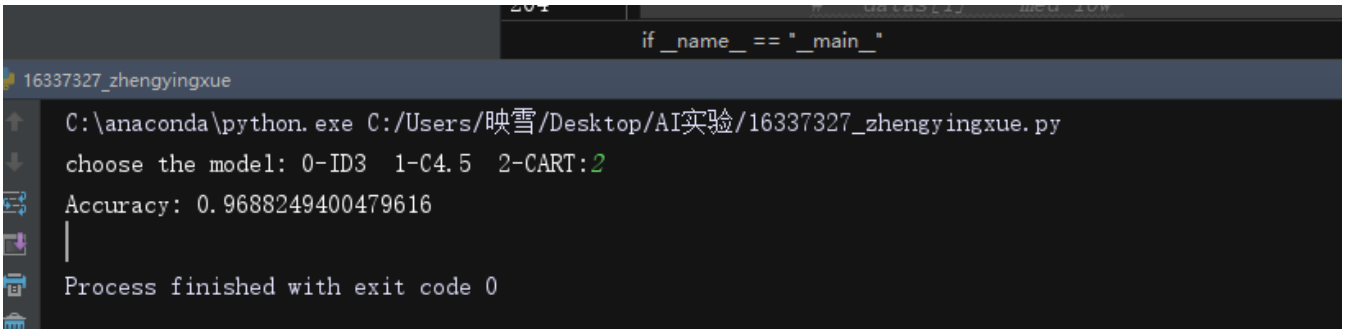


对于 CART 模型，我进行了两次尝试，分别是二叉和多叉。

下面是二叉的准确率：



比 ID3 和 C4.5 的准确率小一些，仍然使用多叉 CART 的准确率：



使用多叉 CART 模型与前两个模型的准确率相同，同时比二叉 CART 的准确率高一些，所以最终代码我采用多叉 CART 模型。

思考题

1、决策树有哪些避免过拟合的方法?

答：可以从两个角度出发：

①对于数据集本身，如果噪音数据比较多，那么会出现过拟合的情况。所以我们在选择训练数据时要注意减少噪音数据。

②对于决策树，如果任其自由生长，则会出现过拟合的情况。所以要进行剪枝操作。可以在某种情况下提前终止树的生长，也可以在树建立完毕后进行剪枝。也可以通过建立随机森林来避免过拟合。

2、C4.5 相比于 ID3 的优点是什么，C4.5 又可能有什么缺点?

答：ID3 利用信息增益来选择最优属性，但实际上它偏向于分支较多的属性。C4.5 使用信息增益率的概念，使每个属性的信息增益除以它们本身的熵，这样一来就做了一个惩

罚，因为分支较多的属性，它本身的熵就比较大。使用信息增益率来选择最优属性可以一定程度上规避 ID3 的偏向性。

C4.5 本身的缺点是需要进行的运算比较多，当数据量比较大的时候，建树需要进行大量的运算，从而提升了算法的复杂性。

3、如何用决策树来进行特征选择（判断特征的重要性）？

答：此题实际上是决策树建树方法的一个总结。

对于 ID3 决策树：使用信息增益作为最优属性的选择标准。信息增益越大，则信息纯度越高。因此，当遍历完所有特征的时候，特征的重要性与信息增益的大小正相关，选取最优属性则选取信息增益最大的属性即可。

对于 C4.5 决策树：一定程度上可以克服 ID3 偏向于分支较多属性的缺点，使用信息增益率作为最优属性的评判标准，对分支较多的属性进行一定的惩罚。特征的重要性与信息增益率的大小正相关，选取最优属性则选取信息增益率最大的属性即可。

对于 CART 决策树：CART 决策树多为二叉树，本实验为多叉树。使用 GINI 系数作为最优属性的选择标准。GINI 系数代表信息的不纯度，所以 GINI 属性越小，信息纯度越高。因此，特征的重要性与 GINI 系数负相关，选取最优属性则选取 GINI 系数最小的属性即可。